

Bearbeiter: Zottmann, Harald

Betreuer: Dipl.-Ing. W. Ruppel

Entwicklung von Programmen zur Simulation eines PALplus-Decoders unter KHOROS.

Die Aufgabe dieser Diplomarbeit umfaßt die Entwicklung von Programmen zur Simulation eines PAL/PALplus-Decoders.

Das Simulationspaket für digital aufgezeichnete PAL/PALplus-Signale von einer D3 Bandmaschine zur späteren Ausgabe auf Standard 4:3 und 16:9 Monitoren umfaßt zehn eigenständige Module. Das modulare Konzept und ein genormtes Dateiformat zum Datenaustausch zwischen den einzelnen Funktionsgruppen ermöglicht eine flexible Ergänzung des Simulationspakets.

Die Simulationsumgebung ist KHOROS in der Programmiersprache „C“ unter UNIX.

Inhaltsverzeichnis

1	Einleitung	5
2	Der PAL-Standard	6
2.1	PAL-Coder	6
2.2	PAL-Decoder	7
3	Empfängerseitige Maßnahmen	9
3.1	Trennung von Chrominanz und Luminanz	9
3.1.1	Grundlagen der Kammfilter	11
3.1.2	Zeilen- und Bildkammfilter	11
3.1.3	Adaptive Zeilen- und Bildkammfilter	13
3.1.4	Bewegungsdetektion zur adaptiven Umschaltung	13
3.2	Verbesserung der subjektiven Bildschärfe	14
3.2.1	Peaking	15
3.2.2	Color Transient Improvement (CTI)	16
3.3	Medianfilter zur Rauschunterdrückung	18
4	PALplus	21
4.1	Das Letterbox-Format	21
4.2	COLORplus	22
4.2.1	COLORplus-Coder	27
4.2.2	COLORplus-Decoder	28
4.2.3	Bewegungsdetektion für COLORplus	28
4.2.4	Beispiele zur Bewegungsdetektion	31
4.3	PALplus-Codec	34
4.4	Vertikale Bandaufspaltung	36
4.4.1	Quadratur-Spiegelfilter (QMF)	36
4.4.2	Helper-Signalverarbeitung	37
4.5	Daten-Bits in Zeile 23	38
4.5.1	Codierung der Daten-Bits	38
4.5.2	Fehlererkennung mit dem CRC-8 Polynom	39
4.5.3	Bedeutung der Daten-Bits	40
4.5.4	Vorschriften zur Arbeitsweise des Decoders	41
5	Der PALplus-Decoder unter KHOROS	43
5.1	Funktionsgruppen des Decoders	43
5.1.1	raw2cvbs	44
5.1.2	cvbs2yc	47
5.1.3	c2uv	48
5.1.4	yuv2rgb	49
5.1.5	peaking	50
5.1.6	cti	51
5.1.7	medfil	52
5.1.8	colplus	53
5.1.9	line23	56

5.1.10	helper	57
5.2	PAL-Workspace	59
5.3	PALplus-Workspace	60
6	Zusammenfassung	62
A	Programmlistings	67
A.1	raw2cvbs	67
A.1.1	raw2cvbs.pane	67
A.1.2	raw2cvbs.prog	67
A.2	cvbs2yc	76
A.2.1	cvbs2yc.pane	76
A.2.2	cvbs2yc.prog	77
A.3	c2uv	88
A.3.1	c2uv.pane	88
A.3.2	c2uv.prog	89
A.4	yuv2rgb	98
A.4.1	yuv2rgb.pane	98
A.4.2	yuv2rgb.prog	98
A.5	peaking	103
A.5.1	peaking.pane	103
A.5.2	peaking.prog	103
A.6	cti	109
A.6.1	cti.pane	109
A.6.2	cti.prog	110
A.7	medfil	118
A.7.1	medfil.pane	118
A.7.2	medfil.prog	118
A.8	colplus	125
A.8.1	colplus.pane	125
A.8.2	colplus.prog	125
A.9	line23	144
A.9.1	line23.pane	144
A.9.2	line23.prog	144
A.10	helper	158
A.10.1	helper.pane	158
A.10.2	helper.prog	158

Abbildungsverzeichnis

1	PAL-Coder	7
2	PAL-Decoder	8
3	Amplitudenspektrum des CVBS-Signals	9
4	Verkämmung des Luminanz- und Chrominanzspektrums	10
5	Farbträgerfalle und Chrominanzbandpaß	10
6	Kammfilter mit zugehörigem Frequenzgang	11
7	Kammfilter mit Bandpaß	12
8	Kammfilterwirkung im Chrominanz-Zweig	12
9	Kammfilterwirkung im Luminanz-Zweig	12
10	Adaptives Kammfilter	13
11	Transversales Filter zur Bewegungsdetektion	14
12	Verarbeitung des Signals der Bewegungsdetektion	14
13	Schaltung zur horizontalen und vertikalen Aperturkorrektur	15
14	Prinzip der Sprungdetektion	16
15	Beispiele von Polynomen für das CTI-Verfahren	17
16	Blockstruktur für das CTI-Verfahren	18
17	Medianfilter erster Ordnung	18
18	Eingangs- und Ausgangsfolge eines Medianfilters	19
19	Dreidimensionales Medianfenster	20
20	Prinzip der Letterbox-Signalverarbeitung	21
21	Phasenlagen des Farbträgers beim PAL-Verfahren	22
22	Digitaler Halbbildaddierer mit Halbbildspeicher	23
23	Verfügbare Frequenzbereich bei COLORplus	26
24	Frequenzgang des Halbbildaddierers	26
25	COLORplus-Coder mit Bewegungsdetektion	27
26	COLORplus-Decoder mit Bewegungsdetektion	28
27	Matrix zur Bewegungsdetektion bei COLORplus	29
28	Schaltung zur Bewegungsdetektion bei COLORplus	30
29	Testbild für die Bewegungsdetektion in COLORplus	31
30	Bewegungssignal für eine Matrix der Dimension 2x2	31
31	Bewegungssignal für eine Matrix der Dimension 4x4	32
32	Bewegungssignal für eine Matrix der Dimension 6x6	32
33	Bewegungssignal für eine Matrix der Dimension 10x10	33
34	Bewegungssignal nach dem Schwellwertentscheider (10x10 Matrix)	33
35	PALplus-Coder	34
36	PALplus-Decoder	35
37	Filterbank zur Rasterkonversion	36
38	Hoch- und Tiefpaßfilter zur Bandaufspaltung 3:4, 1:4	37
39	Nichtlineare Kennlinien im Coder und Decoder	38
40	Biphase-L Codierung	39
41	Schaltung zur Polynomdivision durch das CRC-8 Polynom	40
42	Programmoberfläche (GUI) von „raw2cvbs“	44
43	Ermitteln der Farbträgerphase	45
44	Flußdiagramm von „raw2cvbs“	46

45	Programmoberfläche (GUI) von „cvbs2yc“	47
46	Programmoberfläche (GUI) von „c2uv“	48
47	Programmoberfläche (GUI) von „yuv2rgb“	49
48	Programmoberfläche (GUI) von „peaking“	50
49	Programmoberfläche (GUI) von „cti“	51
50	Programmoberfläche (GUI) von „medfil“	52
51	Programmoberfläche (GUI) von „colplus“	53
52	Programmoberfläche (GUI) von „line23“	56
53	Programmoberfläche (GUI) von „helper“	57
54	Workspace des PAL-Decoders	59
55	Workspace des PALplus-Decoders	60

1 Einleitung

Die weltweiten Anstrengungen zur Verbesserung der Bildqualität bei der Farbfernsehübertragung führten zu unterschiedlichen Systemvorschlägen in den USA, Japan und Europa. Allen gemeinsam ist jedoch die Änderung des Bildseitenverhältnisses von 4:3 auf 16:9.

Die Einführung des digitalen Fernsehens wird wahrscheinlich erst Ende der 90er Jahre einsetzen. Die bis dahin verbleibende Zeit kann für Verbesserungen der bestehenden Fernsehsysteme, die als Zwischenschritt auf dem Weg zum digitalen Fernsehen zu verstehen sind, genutzt werden. In Europa ist zu diesem Zweck das PALplus-Projekt gegründet worden. Seine Zielsetzung umfaßt [18] [19] [39]:

- Option auf das Bildformat 16:9.
- Erhöhte Bildqualität.
 1. Vermeidung von Cross-Color.
 2. Vermeidung von Cross-Luminanz.
 3. Ausnutzung der Kanalbandbreite von 5.6 MHz für die Luminanz.
 4. Erhöhung der Farbbandbreite.

All diese Forderungen müssen jedoch unter Hinblick auf vollständige Kompatibilität zu dem vorhandenen PAL-Standard (Bandbreite des ausgesendeten Signals und Farbcodierung) verwirklicht werden. Damit wird gewährleistet, daß die vorhandenen PAL-Empfänger ohne nennenswerte Einschränkungen auch weiterhin genutzt werden können. Die Merkmale von PALplus sind dahingegen nur mit Empfängern nutzbar, die mit einem PALplus-Decoder und entsprechend einer 16:9 Bildröhre ausgerüstet sind. Die spätere Umrüstung der neuen PALplus-Empfänger auf digitales Fernsehen sollte sich durch das Austauschen des Decoders einfach und schnell durchführen lassen. Somit kann schon vor der Einführung ein beträchtlicher Markt für das digitale Fernsehen geschaffen werden, wenn PALplus die entsprechende Akzeptanz beim Verbraucher findet.

Gegenstand der vorliegenden Arbeit ist es, Programme zur Simulation eines PAL-Decoders und eines PALplus-Decoders unter der Simulationsumgebung KHOROS zu erstellen. Zu diesem Zweck wurde der PAL/PALplus-Decoder in einzelne Funktionsgruppen unterteilt, die getrennt realisiert werden konnten.

Es folgt zunächst eine kurze Beschreibung des PAL-Standards. Zur Verbesserung der Bildqualität bei Standard-PAL werden in Kapitel 3 einige Algorithmen vorgestellt, die im digitalen Empfänger eingesetzt werden können. In Kapitel 4 wird das Konzept von PALplus dargelegt und erklärt. Die Beschreibung der Programme zur Simulation der einzelnen Algorithmen befindet sich in Kapitel 5. Den Abschluß der Arbeit bildet die Zusammenfassung in Kapitel 6 mit einer Übersicht über alle Algorithmen und Programme.

2 Der PAL-Standard

Die Definition des PAL-Standards lässt sich an Hand des Aufbaus von Coder und Decoder beschreiben. Der Zweck dieser Baugruppen ist der, die Farbvalenzen Rot, Grün und Blau (RGB) über einen einzigen Kanal mit der Bandbreite des Leuchtdichtesignals zu übertragen. Dies ist jedoch nur möglich, weil das menschliche Auge ein geringeres Auflösungsvermögen für farbige Bilddetails hat, als für Luminanzdetails. Deswegen wird nur die Luminanz (Y) mit der vollen Bandbreite übertragen. Die Farbdifferenzsignale (U/V) werden dagegen vor der Übertragung bandbegrenzt. Die Umrechnung von RGB in YUV geschieht über folgendes Gleichungssystem [15]:

$$\begin{pmatrix} Y \\ U = 0.493 \cdot (B - Y) \\ V = 0.877 \cdot (R - Y) \end{pmatrix} = \begin{pmatrix} 0.299 \cdot R & +0.587 \cdot G & +0.114 \cdot B \\ -0.147 \cdot R & -0.289 \cdot G & +0.437 \cdot B \\ 0.615 \cdot R & -0.515 \cdot G & -0.100 \cdot B \end{pmatrix} \quad (1)$$

Die Decoder-Matrix zur Umrechnung von YUV in RGB ergibt sich daraus zu:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.000 \cdot Y & +0.000 \cdot U & +1.140 \cdot V \\ 1.000 \cdot Y & -0.394 \cdot U & -0.581 \cdot V \\ 0.998 \cdot Y & +2.028 \cdot U & +0.000 \cdot V \end{pmatrix} \quad (2)$$

Die Übertragung der Helligkeit in einem Kanal, und der Farbdifferenzsignale in anderen Kanälen, nennt man das „Prinzip der konstanten Helligkeit“ („Constant-Luminance“-System). Hierdurch ist die empfängerseitige Kompatibilität zu den bestehenden Schwarzweiß-Empfängern gegeben. Weiterhin erreicht man dadurch eine geringe Störbeeinflussung des Luminanzanteils bei der Übertragung.

2.1 PAL-Coder

Die Schaltung des PAL-Coders [30] in Abbildung 2.1 beginnt mit der Coder-Matrix, welche die Farbvalenzen RGB in die zur Übertragung besser geeigneten YUV-Komponenten matriziert. Die beiden Farbdifferenzsignale (U/V) werden auf 1.5 MHz bandbegrenzt. Dem U-Modulator wird der Farbträger mit einer Phasenlage von 0° zugeführt, während der V-Modulator einen von Zeile zu Zeile alternierenden Farbträger mit einer Phasenlage von $\pm 90^\circ$ erhält. Nach der Modulation werden die trägerfrequenten U/V-Komponenten addiert. Es wird also eine Quadratur-Amplitudenmodulation des Farbträgers durchgeführt, wobei die Information über den Farbton in der Phase und die Sättigungsinformation in der Amplitude des modulierten Farbträgers enthalten ist. Über die vor den Modulatoren gelegenen Addierstufen wird der Burst-Tastimpuls auf die hintere Schwarzscher des CVBS-Signals aufgetastet.

Somit liefern die Modulatoren einen zeilenweise phasenalternierenden Burst mit einer Phasenlage von $\pm 135^\circ$. Die Phasenmodulation des Burstes wird im Decoder benötigt, um einen synchronen Lauf zwischen Coder- und Decoderschaltphase zu gewährleisten.

Im Luminanzkanal wird das Helligkeitssignal zunächst von Frequenzanteilen befreit, die in der Nähe der Farbträgerfrequenz liegen. Dadurch wird verhindert, daß Rauschanteile des Luminanzsignals in den Farbkanal des Decoders gelangen (Cross-Noise). Weiterhin wird mit dieser Maßnahme auch das Übersprechen der farbträger-nahen Luminanz in die Chrominanz (Cross-Color) reduziert. Die Austastschaltung versieht alle horizontalen und vertikalen Austastlücken mit definierten Pegeln. Da der Farbkanal wegen der kleineren Bandbreite seiner Filter eine längere Laufzeit aufweist als das Luminanzsignal, muß es noch entsprechend verzögert werden.

Abschließend werden die Signale addiert und mit der nötigen Synchroninformation vom Studio, die um die gesamte Coderlaufzeit verzögert wird, versehen.

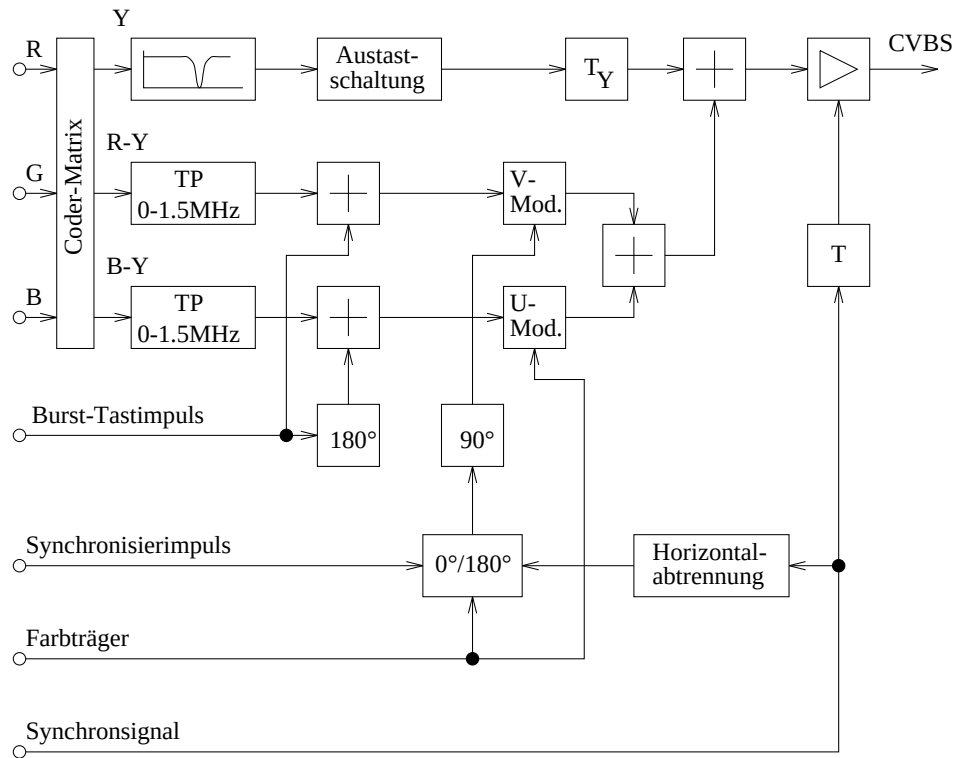


Abbildung 1: PAL-Coder

2.2 PAL-Decoder

In Abbildung 2.2 ist das Schaltbild eines Standard-PAL-Decoders [30] aufgezeigt. Zuerst wird das CVBS-Signal nach Luminanz- und Chrominanzanteil aufgespalten. Die Frequenzgänge der hierfür benötigten Filter sind in Abbildung 3.1 dargestellt. Der modulierte Farbträger, der die Chrominanzinformation trägt, wird in einer PAL-Verzögerungsleitung in die trägerfrequenten Farbsignalkomponenten V_F und U_F zer-

legt. Die PAL-Verzögerungsleitung bewirkt eine vektorielle Addition zweier aufeinanderfolgender Zeilen, um aus Phasenfehlern des Farbträgers resultierende Farbtonverschiebungen in harmlosere Sättigungsverluste zu überführen. Die Demodulation erfolgt dann in zwei getrennten Synchrondemodulatoren. Der V-Demodulator wird dazu mit einem zeilenweise um $\pm 90^\circ$ geschalteten Farbträger gespeist, um den Phasensprung der V-Komponente aufzuheben.

Beim Simple-PAL Verfahren wird die Luminanz mit einem Tiefpaß bei 3 MHz abgeschnitten. Die Demodulation der Chrominanz erfolgt direkt aus dem bandpaßgefilterten CVBS-Signal. Bei Phasenfehlern des Farbträgers wird die zeitliche und geometrische Integrationswirkung des Auges genutzt, um die vektorielle Addition der PAL-Verzögerungsleitung zu ersetzen.

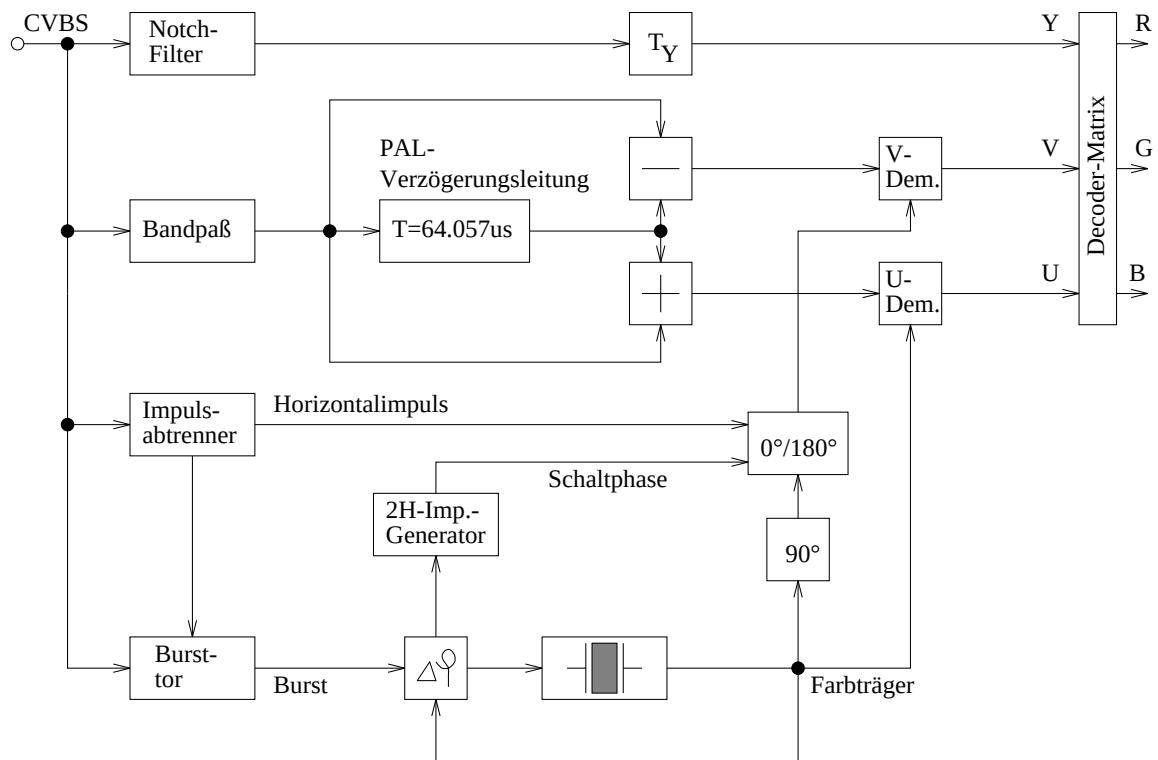


Abbildung 2: PAL-Decoder

Zur Farbträgerückgewinnung wird der Burst aus dem CVBS-Signal herausgetastet und einem Nachlaufsynchronisator mit Quarzoszillator und Phasenbrücke zugeführt. Der Burst wird an der Phasenbrücke mit dem vom Quarzoszillator regenerierten Farbträger verglichen. Wenn die Burstphasenlage nicht gleich der des Farbträgers ist, gibt die Phasenbrücke einen Synchronisierimpuls ab. Dieser, jede zweite Zeile auftretende, Impuls enthält die Information über die Schaltphase des Coders und wird zur Synchronisierung genutzt.

Die demodulierten U- und V-Komponenten werden schließlich noch tiefpaßgefiltert und können dann über die Decoder-Matrix in die zur Ansteuerung des Bildschirms nötigen RGB-Signale umgerechnet werden.

3 Empfängerseitige Maßnahmen

Der Übergang zur digitalen Signalverarbeitung in der Videotechnik [31] [32] erlaubt vielfältige Möglichkeiten zur Bildverbesserung auf der Seite der Wiedergabegeräte. So sind durch die Verwendung von Zeilen- und Bildspeichern neuartige Filterstrukturen möglich geworden, die vielfältig eingesetzt werden können. Maßnahmen dieser Art, die sich lediglich auf den Empfänger beschränken, können allgemein unter dem Begriff IDTV (= Improved Definition Television) zusammengefasst, obwohl diese Techniken zur Verbesserung der Bildqualität nicht unbedingt auch eine Verbesserung der Auflösung bewirken.

3.1 Trennung von Chrominanz und Luminanz

Nach dem PAL-Standard wird die tiefpaßgefilterte Chrominanz geträgert (Farbträgermodulation mit Trägerunterdrückung) im Luminanzband übertragen (Frequenzmultiplex). Die Frequenz des Farbträgers wurde so gewählt, daß sich die Amplitudenspektren von Luminanz und Chrominanz (U/V-Komponente) ineinander verkämmen. Dies erreicht man durch die Verkopplung der Farbträgerfrequenz f_{sc} mit der Zeilenfrequenz f_h . Die Gleichung zur Wahl der Farbträgerfrequenz heißt demnach:

$$f_{sc} = \left(n - \frac{1}{4}\right) \cdot f_h + 25 \text{ Hz} = 4.433\,618\,75 \text{ MHz} \quad (3)$$

mit $n = 284$ und $f_h = 15\,625 \text{ Hz}$.

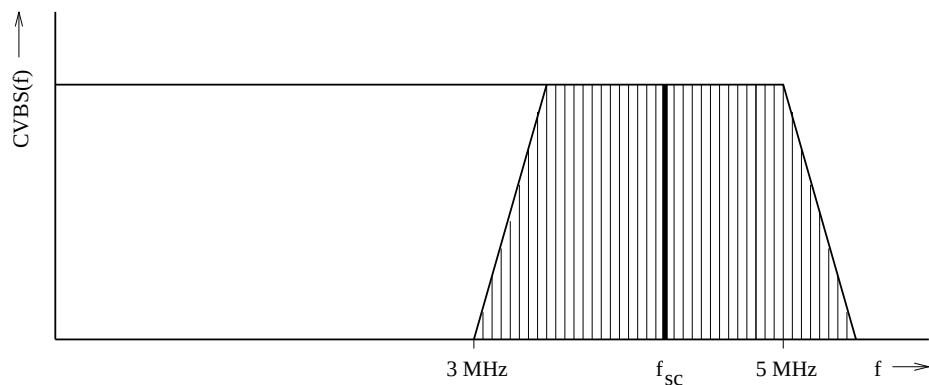


Abbildung 3: Amplitudenspektrum des CVBS-Signals

Bedingt durch die Wahl der Farbträgerfrequenz sind die Positionen der U-Komponente um eine Viertelfrequenzperiode bezüglich der zugehörigen Y-Frequenzlinie verschoben. Die zeilenweise Umschaltung der V-Komponente (PAL = Phase Alternating Line) bewirkt eine zusätzliche Modulation, welche die U- und V-Komponente um eine halbe Frequenzperiode voneinander trennt. Ein Bild mit rein vertikalen Strukturen würde demnach nur diskrete Frequenzlinien nach Abbildung 3.1 hervorrufen. Bei unbewegten Bildern mit beliebigem Inhalt wiederholt sich diese Frequenzstruktur im Abstand der Vollbildfrequenz.



Abbildung 4: Verkämmung des Luminanz- und Chrominanzspektrums

Durch den Einsatz eines geeigneten Kammfilters ist es somit prinzipiell möglich, bei unbewegtem Bildinhalt eine fast perfekte Trennung der YUV-Frequenzkomponenten bei der Demodulation durchzuführen. Bei bewegtem Bildinhalt rücken die Frequenzlinien unter Umständen jedoch soweit zusammen, daß eine übersprechfreie Trennung nicht mehr zu bewerkstelligen ist.

Kammfilter konnten in analoger Technik bisher nur sinnvoll unter Verwendung von teuren Verzögerungsleitungen realisiert werden. Die meisten Gerätehersteller sind deswegen zu einer kostengünstigeren Methode der Y/C-Trennung übergegangen. Die Lage des Farbträgers im Luminanzkanal, nahe der oberen Bandgrenze, ermöglicht die Trennung von Chrominanz und Luminanz durch ein Notch-Filter, mit einer Einsattelung bei der Farbträgerfrequenz, im Luminanzkanal, und einem zur Farbträgerfrequenz symmetrischen Bandpaß im Chrominanzkanal [30] (Abbildung 3.1). Durch dieses Verfahren erreicht man für gewöhnlich jedoch lediglich eine Luminanzbandbreite von etwa 3.5 MHz.

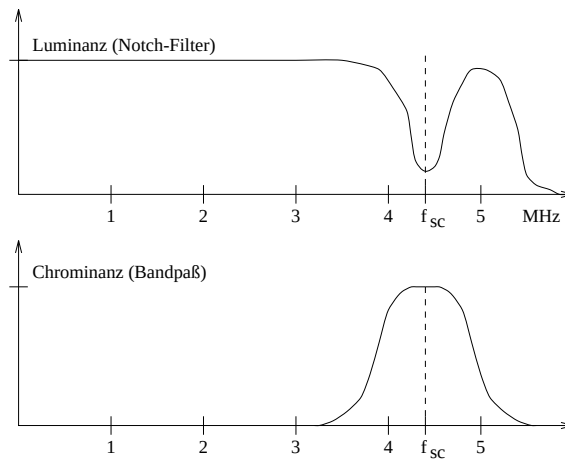


Abbildung 5: Farbträgerfalle und Chrominanzbandpaß

Weitere Nachteile dieses Verfahrens sind das Übersprechen der hochfrequenten Luminanz in die Chrominanz, was zu farbigen Cross-Color-Störungen führt, sowie das Übersprechen der Chrominanz in die Luminanz (Cross-Luminanz) beim Übergang von Farben hoher Sättigung, was als störende Luminanzmuster in Erscheinung tritt. Solche eindimensionalen Filter können auch in der Digitaltechnik realisiert werden. Ihr Vorteil liegt dann in der phasenlinearen Filtercharakteristik, sowie in

der Langzeitstabilität. Die eigentlichen Vorteile der Digitaltechnik erschließen sich aber erst mit der Nutzung von vertikal oder zeitlich wirkenden Filteranordnungen. Diese mehrdimensionalen Filter lassen sich einfach unter Verwendung von Speicherbausteinen aufbauen.

3.1.1 Grundlagen der Kammfilter

Die Grundlage der Kammfilter bilden in der Analogtechnik die Verzögerungsleitungen. In der Digitaltechnik läßt sich die Funktion von Verzögerungsleitungen durch Speicherbausteine realisieren. Abbildung 3.1.1 zeigt zwei Grundstrukturen mit Kammfilterwirkung. Das Eingangssignal wird aufgespalten, einmal um T verzögert und einmal direkt auf eine Subtraktions- bzw. Additionsstufe gegeben.

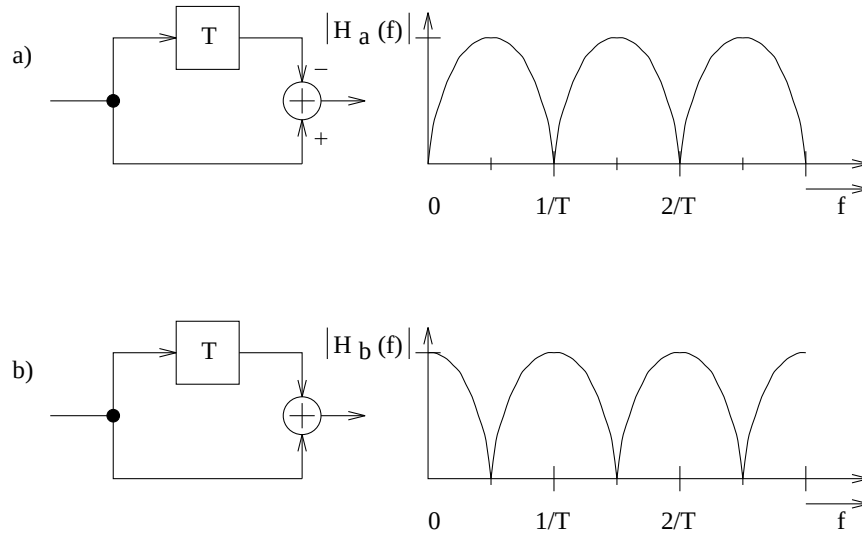


Abbildung 6: Kammfilter mit zugehörigem Frequenzgang

Der Frequenzgang für die erste Anordnung ist damit $H_a(w) = 1 - e^{-j\omega T}$, während die zweite Anordnung den Frequenzgang $H_b(w) = 1 + e^{-j\omega T}$ aufweist. Mit diesen Grundstrukturen können nun beliebige Filter mit äquidistanten Nullstellen und Maxima aufgebaut werden.

3.1.2 Zeilen- und Bildkammfilter

Abbildung 3.1.2 zeigt eine zur Trennung von Chrominanz und Luminanz geeignete symmetrische Filterstruktur für PAL-Signale [32]. Dieses transversale Filter weist jeweils im Abstand der halben Zeilenfrequenz Nullstellen und Maxima auf, wenn die Verzögerungszeiten zwischen den einzelnen Filterzweigen zwei Zeilenperioden betragen. Bei einer Verzögerungszeit von zwei Vollbildern ist der Frequenzgang periodisch mit der halben Bildfrequenz. Die Filterstruktur führt zunächst zu einem Hochpaßverhalten, mit dem die Chrominanz abgetrennt wird. Der Bandpass begrenzt diese Wirkung auf den Frequenzbereich, in dem die Luminanz und die modulierte Chrominanz gemeinsam vorkommen.

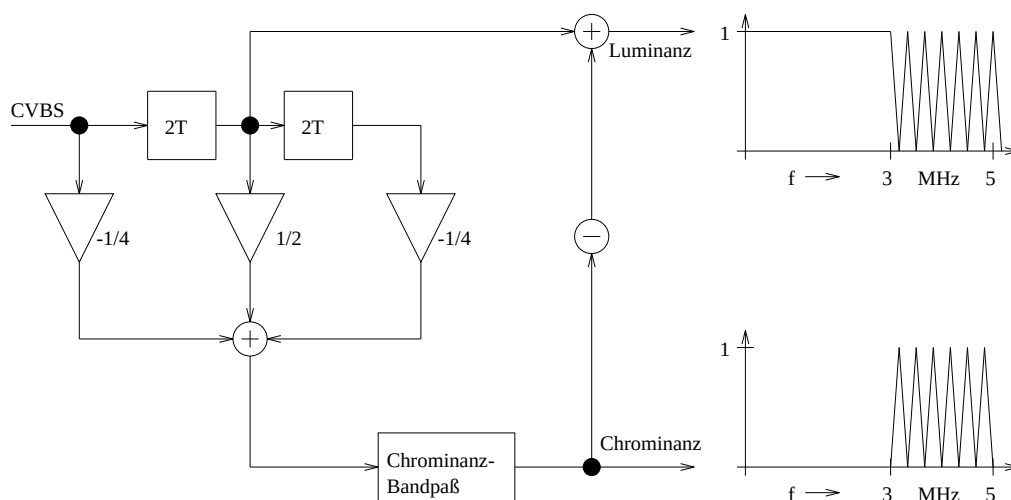


Abbildung 7: Kammfilter mit Bandpaß

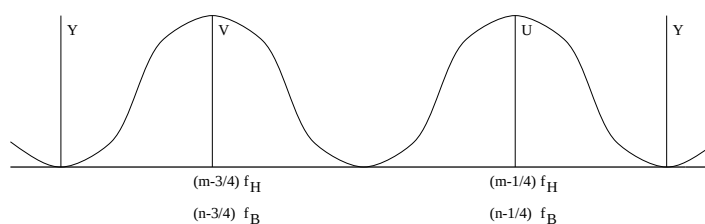


Abbildung 8: Kammfilterwirkung im Chrominanz-Zweig

Die Addition des resultierenden negierten Chrominanzsignals zum laufzeitangepaßten CVBS-Signal, liefert im Luminanzkanal den inversen Frequenzgang zum Chrominanzkanal.

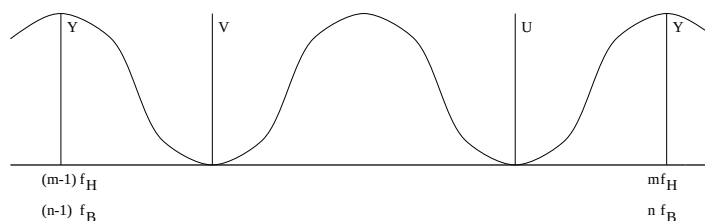


Abbildung 9: Kammfilterwirkung im Luminanz-Zweig

Durch die Verwendung von Bildkammfiltern bleiben alle unbewegten Bilder frei von Störungen. Bei bewegten Szenen müssen Zeilenkammfilter verwendet werden. Diese können alle vertikal strukturierten Bildinhalte mit der vollen Auflösung von 5 MHz wiedergeben. Cross-Color kann bei Zeilenkammfiltern nur an feinen diagonalen Strukturen festgestellt werden. Bei Bildkammfiltern tritt Cross-Color nur in sich bewegenden Luminanzstrukturen auf. Der Nachteil dieser Filter liegt in der Entstehung von Cross-Luminanz-Störungen an horizontalen und an bewegten Farbkanten.

3.1.3 Adaptive Zeilen- und Bildkammfilter

Die Tatsache, daß das Bildkammfilter nur für unbewegte Bilder geeignet ist, führt zu der Überlegung, die Vorteile von Bild- und Zeilenkammfilter in einer adaptiv umschaltenden Filteranordnung zu kombinieren. Abbildung 3.1.3 zeigt eine hierfür geeignete Filterstruktur [10] [36] [37].

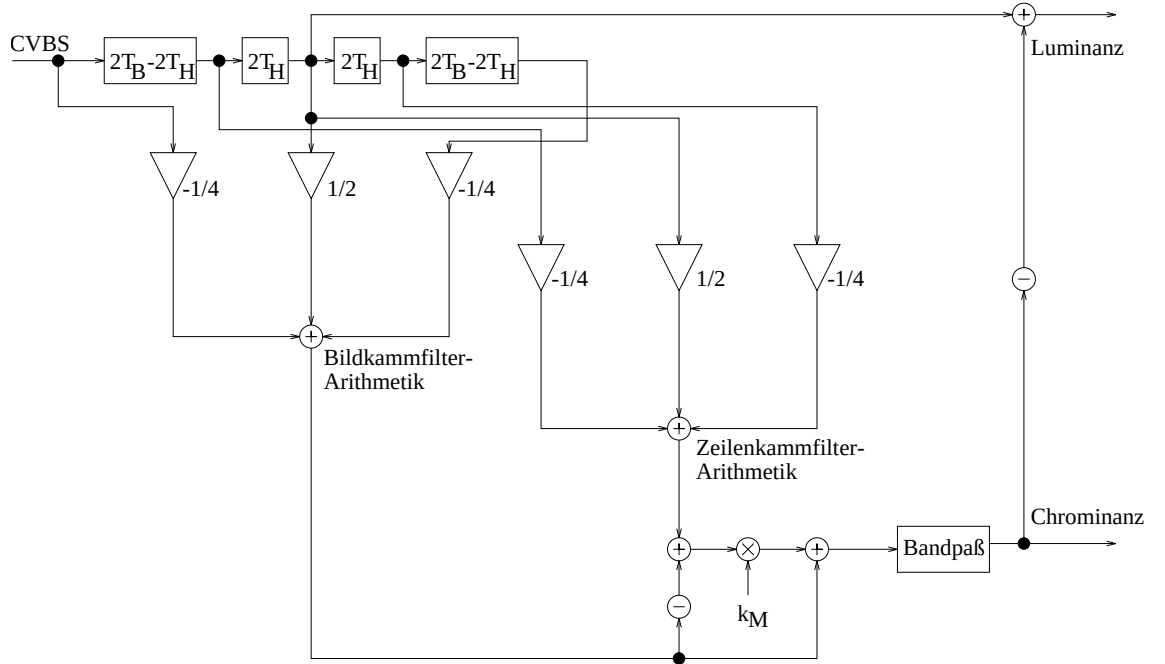


Abbildung 10: Adaptives Kammfilter

Die laufzeitgleichen Signale der Bild- und Zeilenkammfilter-Arithmetik werden einem Mischer zugeführt, der in Abhängigkeit von der Größe k_M zwischen beiden Signalen überblendet. Wenn keine Bewegung in der betreffenden Bildpartie festgestellt wurde ist $k_M = 0$ und es wirkt nur die zeitliche Filterung des Bildkammfilters. Mit zunehmender Bewegung steigt k_M zu immer größeren Werten, bis mit $k_M = 1$ nur noch das Zeilenkammfilter zum Ausgangssignal beiträgt.

3.1.4 Bewegungsdetektion zur adaptiven Umschaltung

Die Überblendung zwischen Bild- und Zeilenkammfilter erfordert ein Steuersignal, welches bei bewegten Bildinhalten große Werte annimmt und bei unbewegten Bildinhalten gleich Null ist. Das Grundprinzip der Bewegungsdetektion beruht auf der Differenzbildung von aufeinanderfolgenden Vollbildern. Da beim PAL-System jedoch benachbarte Vollbilder eine gegenseitige Phasenverschiebung des Farbträgers um 90° aufweisen, muß die Differenzbildung auf eine Periode von 4 Vollbildern ausgedehnt werden. Die hierfür entsprechende Filterstruktur mit dem zugehörigen Frequenzgang [34] zeigt Abbildung 3.1.4.

Das Differenzsignal, das die Information über sich ändernde Bildausschnitte trägt, kann in dieser Form noch nicht zur Überblendung genutzt werden. Der Grund

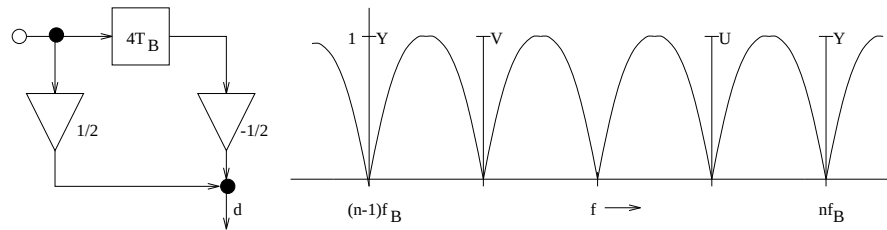


Abbildung 11: Transversales Filter zur Bewegungskennung

hierfür liegt darin, daß zur Überblendung ein Signal zwischen 0 und 1 benötigt wird. Das Signal am Filterausgang ist jedoch vorzeichenbehaftet. Der erste Schritt zur Weiterverarbeitung beinhaltet demnach die Betragsbildung des Steuersignals. Weiterhin stellt sich das Problem, daß auch Rauschanteile im Komponentensignal zu Ausgangssignalen des Filters führen. Eine horizontale und vertikale Mittelwertbildung verhindert dies und ermöglicht eine Glättung des Ausgangssignals. Schließlich wird das Steuersignal mit einer nichtlinearen Kennlinie bewertet, die die Art der Überblendung von weich bis hart beeinflusst und für ausreichende Pegel im Steuersignal sorgt. Abbildung 3.1.4 zeigt den Funktionsblock zur Verarbeitung des Steuersignals noch einmal anschaulich.

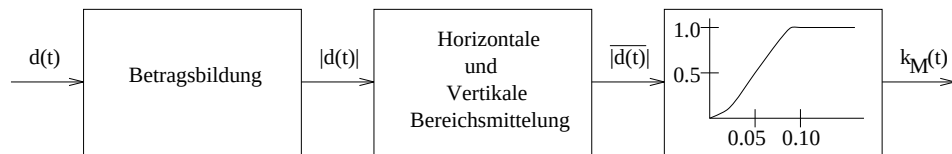


Abbildung 12: Verarbeitung des Signals der Bewegungskennung

Eine andere Möglichkeit der Gewinnung eines solchen Steuersignals erfordert die vorherige Demodulation der YUV-Komponenten. Aus den Komponentensignalen läßt sich dann auch im Vollbildabstand eine Differenzbildung durchführen. Dieses Verfahren hat den Vorteil des geringen Speicherbedarfs, wobei jedoch ein zusätzlicher Hardwareaufwand durch den benötigten PAL-Decoder vor der Bewegungserkennung entsteht.

3.2 Verbesserung der subjektiven Bildschärfe

Ein Kriterium für die Güte eines Fernsehbildes ist sicherlich die Schärfe des Bildes. Die Definition dieser Bildschärfe erfolgt über die Modulationstiefe bei 5 MHz. Sie wird gemessen, indem man die Amplitude einer 5 MHz-Schwingung in Beziehung zur Amplitude einer niederfrequenten Schwingung setzt. Dabei muß jedoch der Verlauf des Frequenzgangs bekannt sein. Eine allgemeinere Meßmethode besteht darin, die Anstiegszeit und das Überspringen eines Schwarzweißsprungs zu messen.

Bestimmt wird die Modulationstiefe durch die Apertur der Fernsehkamera sowie den Frequenzgang der einzelnen Verstärker und Übertragungssysteme. Man unterscheidet zwischen horizontaler Auflösung, die von der Kamera und dem elektrischen

Übertragungssystem abhängig ist, und der vertikalen Auflösung, die nur durch die Apertur der Kamera beeinträchtigt wird. Bei reinen Aperturfehlern handelt es sich demnach lediglich um Amplitudenfehler, die mit keinem Phasenfehler verbunden sind. Bei ihrer Korrektur muß also auch die Anhebung des Frequenzgangs phasenlinear erfolgen. Aus der Analogtechnik sind einige Schaltungen zur Aperturkorrektur bekannt. Nachfolgend werden zwei Schaltungen zur Verbesserung der Bildschärfe vorgestellt, wie sie in der Digitaltechnik verwendet werden können.

3.2.1 Peaking

Das in Abbildung 3.2.1 dargestellte Aperturkorrekturfilter kompensiert den Frequenzgangabfall der Bildaufnahmeeinrichtung, indem es diejenigen Frequenzkomponenten verstärkt, die für den subjektiven Schärfeeindruck des Betrachters maßgebend sind [32].

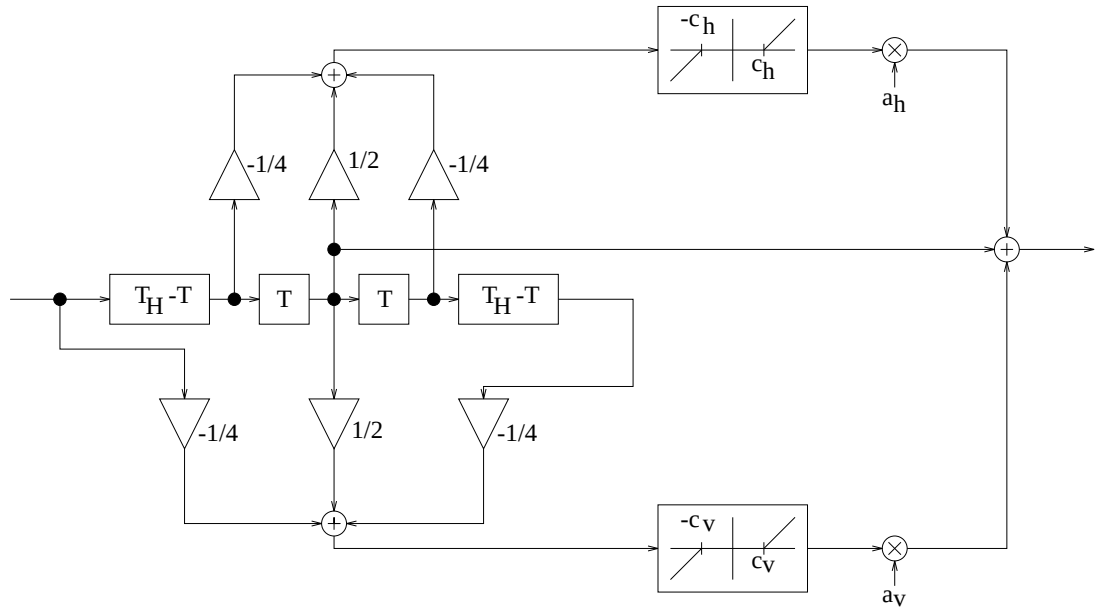


Abbildung 13: Schaltung zur horizontalen und vertikalen Aperturkorrektur

Die Zweige des vertikalen und horizontalen Hochpaßfilters liefern ein Signal über den Detailgehalt des Bildes. Diese Information kann nun kontrolliert dem Originalsignal zugeführt werden. Die Schwellwert-Funktionen in den Zweigen bewirken ein „Coring“ welches verhindert, daß in flächigen Bildpartien enthaltenes Rauschen über den Pegel der Detailsignale angehoben wird. Nachfolgend ist das Bewertungsschema der einzelnen Bildpunkte in einem Koeffizientenarray dargestellt.

$$\begin{array}{ccc}
 0 & \frac{-a_v}{4} & 0 \\
 \frac{-a_h}{4} & 1 + \frac{a_h}{2} + \frac{a_v}{2} & \frac{-a_h}{4} \\
 0 & \frac{-a_v}{4} & 0
 \end{array} \quad (4)$$

Dieser lineare Filteransatz eignet sich nur für die demodulierte Luminanz. Die Bandbreite der U- und V-Komponente ist schon derart stark bandbegrenzt (≈ 1.5 MHz), daß sich mit linearen Techniken keine weiteren Verbesserungen mehr erreichen lassen.

3.2.2 Color Transient Improvement (CTI)

Zur Schärfeverbesserung der bei PAL auf 1.5 MHz bandbegrenzten Farbdifferenzsignale U und V, eignen sich nur nichtlineare Verfahren. Die dem Color Transient Improvement (CTI) zugrunde liegende Idee [6] beruht darauf, Signalsprünge zu versteilern, indem sie durch Polynomfunktionen ersetzt werden. Eine wichtige Forderung für diese Polynome ist, daß sie symmetrisch um den zeitlichen Mittelpunkt der originalen Übergangsfunktion sind. Damit wird die zusätzliche Entstehung von Jitter und der zeitliche Versatz zwischen Luminanz und Chrominanzsignal an den Farbübergängen verhindert.

Die typische Länge eines Übergangsbereichs hängt über die Beziehung $BT = 1$ stark von der Bandbreite des Vorfilters für das Signal ab. Es läßt sich somit eine Mindest- und eine Maximalbandbreite angeben, über die der typische Längenbereich der Übergänge definiert wird, welche durch das CTI-Verfahren bearbeitet werden sollen. Auf diese Weise lassen sich Bereiche mit sich langsam ändernden Farben von der Verarbeitung ausschließen. Auch durch Rauschen hervorgerufene impulsförmige Signalverläufe, die kürzere Anstiegszeiten haben als durch die Bandbegrenzung möglich ist, können somit lokalisiert und unterdrückt werden. Abbildung 3.2.2 zeigt die Vorgehensweise bei der Gewinnung des Steuersignals zur Sprungversteilerung.

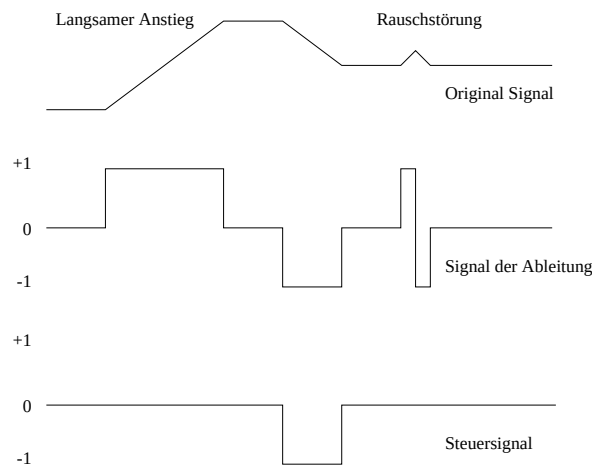


Abbildung 14: Prinzip der Sprungdetektion

Mit Hilfe einer einfachen Differenzenbildung zwischen benachbarten Bildpunkten wird ein Signal gewonnen, das eine Information über die Steigung des Eingangssignals enthält. Dieses Signal benötigt nur drei Pegel (-1, 0, +1), welche die jeweilige Richtung des Gradienten repräsentieren. Das so ermittelte Steuersignal wird dazu benutzt, die Konturen auszuwählen, die versteilert werden sollen. Handelt es sich etwa um einen Sprung mit positiver Steigung, so wird von dem CTI-Algorithmus

zuerst Anfangs- und Endwert des Übergangs ermittelt. Die Differenz der beiden Werte repräsentiert dann die Höhe des Übergangs. Über die Länge des Übergangs wird dann ein auf 1 normiertes Polynom errechnet. Jeder Wert des Polynoms wird mit der Höhe des Originalübergangs multipliziert und anschließend mit dem Anfangswert des Übergangs addiert. Abbildung 3.2.2 zeigt einige Polynome die beim CTI-Verfahren verwendet werden können.

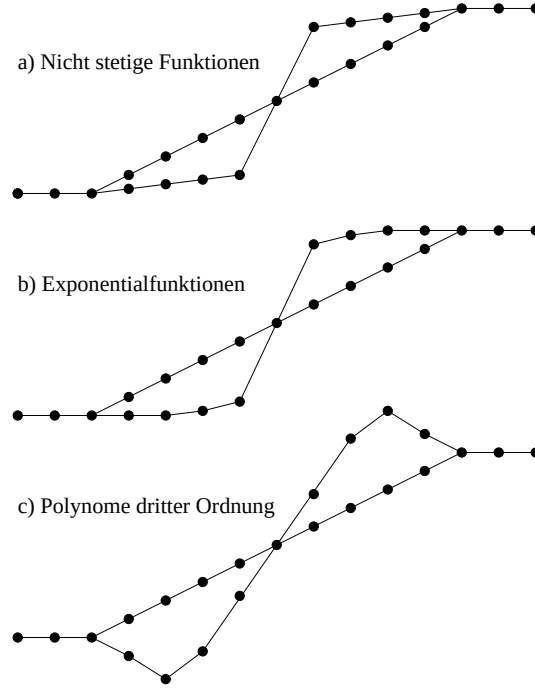


Abbildung 15: Beispiele von Polynomen für das CTI-Verfahren

a) Nicht stetige Funktionen:

$$P(x) = \begin{cases} x^2 & 0 \leq x < \frac{1}{2} \\ 1 - (1 - x)^2 & \frac{1}{2} \leq x \leq 1 \end{cases} \quad (5)$$

b) Exponentialfunktionen:

$$P(x) = \begin{cases} 2^{p-1}x^p & 0 \leq x < \frac{1}{2} \\ 1 - 2^{p-1}(1 - x)^p & \frac{1}{2} \leq x \leq 1 \end{cases} \quad (6)$$

c) Polynome dritter Ordnung:

$$P(x) = \begin{cases} ax^3 + bx & 0 \leq x < \frac{1}{2} \\ 1 - (a(1 - x)^3 + b(1 - x)) & \frac{1}{2} \leq x \leq 1 \end{cases} \quad (7)$$

mit $a = 4 / (1 - 12x_0^2)$ und $b = -3x_0^2$, wobei $0 \leq x_0 \leq \sqrt{12}$. Dabei gilt, daß x_0 bzw. $1 - x_0$ den Punkt des Polynom-Minimums bzw. Maximums definiert.

Das CTI-Verfahren ermöglicht einen deutlichen Gewinn an Schärfe im Vergleich zum gefilterten Originalsignal. Eine weitere positive Eigenschaft des Verfahrens ist,

daß keine wahrnehmbaren Artefakte entstehen. Die Verwendung von Polynomfunktionen, im Gegensatz zu einfachen Sprungfunktionen, vermeidet die Verletzung des Abtasttheorems.

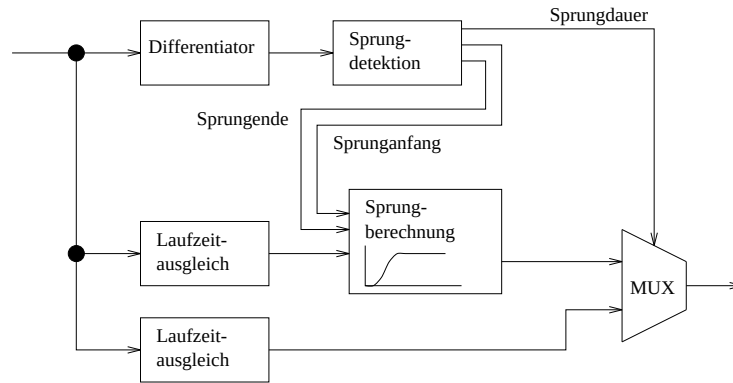


Abbildung 16: Blockstruktur für das CTI-Verfahren

3.3 Medianfilter zur Rauschunterdrückung

In der Videotechnik werden als gebräuchlichste Filtertypen bisher die Transversalfilter und die rekursiven Filter verwendet. Als Vorteile der Transversalfilter ist ihre konstante Gruppenlaufzeit zu nennen. Diese Eigenschaft macht sich gerade in der Videotechnik sehr angenehm bemerkbar, da Laufzeitfehler mit dem menschlichen Auge leicht wahrnehmbar sind. Auch für die rekursiven Filter gibt es, trotz nicht konstanter Gruppenlaufzeit, gerade im Bereich der Rauschreduktion, einige Anwendungsmöglichkeiten.

In diesem Kapitel soll jedoch ein nichtlinearer Filtertyp vorgestellt werden, der bisher in der Videotechnik noch kaum zum Einsatz kommt, und sich in seiner Funktionsweise grundlegend von anderen Filtertypen unterscheidet. Die Medianfilter stellen als Funktion einen Rangordnungsoperator dar [32]. Ein bestimmter Vorrat von Abtastwerten am Eingang wird nach ihrer Amplitudengröße sortiert und der mittlere Wert der sortierten Reihe wird an den Ausgang gegeben. Abbildung 3.3 zeigt das Blockschaltbild eines solchen Medianfilters erster Ordnung [38].

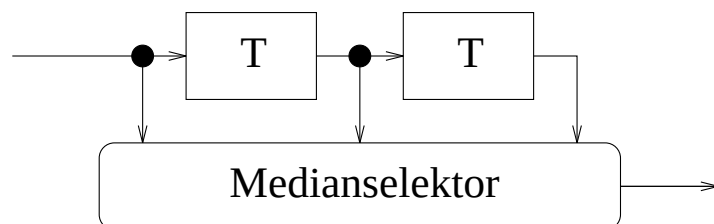


Abbildung 17: Medianfilter erster Ordnung

Die Medianfilterung stellt einen reinen Umsortierungsprozeß dar. Dadurch ent-

stehen keinerlei Quantisierungsfehler. Da das Verfahren stark nichtlinear ist, entstehen jedoch Aliasstörungen.

Beim Medianfilter ist die übliche Beschreibungsart von Filtern im Frequenzbereich nicht möglich. Sein Verhalten im Zeitbereich zeigt aber bemerkenswerte Eigenschaften. So läßt sich feststellen, daß kurze impulsförmige Störspitzen vom Medianfilter vollständig eliminiert werden. Dieses Verhalten entspricht dem eines Tiefpasses. Im Gegensatz zu diesem werden jedoch Sprungfunktionen ohne Verzögerung und in ihrer vollen Höhe durchgelassen. Die Übertragungsfunktion für Impulsfunktionen ist demnach 0, die für Sprungfunktionen ist jedoch 1.

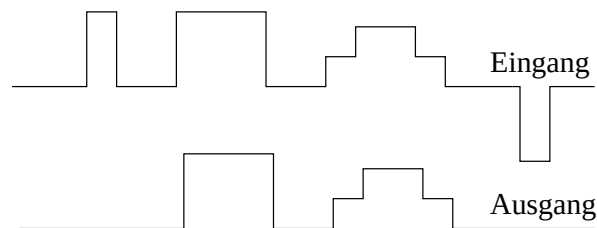


Abbildung 18: Eingangs- und Ausgangsfolge eines Medianfilters

Das Beispiel in Abbildung 3.3 zeigt, daß der Dämpfungsverlauf eines Medianfilters abhängig vom Kurvenverlauf des Eingangssignals ist. Weiterhin ist ersichtlich, daß das Medianfilter sehr gut geeignet ist, um kurze Störimpulse auszufiltern. Wenn man Medianfilter zur Störsignalbefreiung in der Videotechnik einsetzen will, so ist zunächst zu beachten, welche Störungen dabei überhaupt vorkommen können, und in welche Klasse von Störungen sie fallen:

- Störungen, die statistisch unabhängig in den drei Abtastrichtungen sind.
 1. Kamerarauschen.
 2. Rauschen von der Magnetaufzeichnung (MAZ).
 3. Kornrauschen bei der Wiedergabe vom Filmen.
- Störungen, die vorzugsweise in der Abtastrichtung vorkommen.
 1. Dropouts vom Magnetband.
 2. Filmkratzer in vertikaler Richtung.
 3. Schmutz auf dem Film.
- Periodische Störungen.
 1. Cross-Color.
 2. Cross-Luminanz.

Das Ziel beim Einsatz eines Medianfilters sollte es sein, alle diese Störungen möglichst gut auszufiltern. Dazu ist die Struktur aus Abbildung 3.3 so zu modifizieren, daß sie diese Anforderungen erfüllt.

Bedingt durch die räumlich und zeitlich periodische Abtastung bei der Videoaufnahme weist ein Videosignal ein dreidimensionales Frequenzspektrum auf. Dies bedeutet weiterhin, daß Bildpunkte mit hoher gegenseitiger Korrelation in der räumlichen und zeitlichen Nachbarschaft zum aktuellen Bildpunkt existieren. Die Funktionsweise des Medianfilters liegt aber gerade darin, aus Bildpunkten mit sehr ähnlicher Information diejenigen auszuwählen, der einen mittleren Rang zwischen ihnen einnimmt, und damit am wenigsten gestört ist. Je mehr Bildpunkte zu diesem Vergleich herangezogen werden können, desto besser muß demnach die Filterwirkung sein.

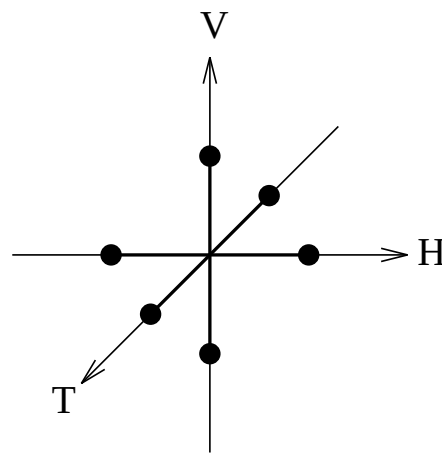


Abbildung 19: Dreidimensionales Medianfenster

Die Ergebnisse mit dieser Filterstruktur (Abbildung 3.3) zeigen, daß sich mit ihr ein beträchtlicher Störabstandsgewinn erzielen läßt. Bei bewegten Bildbereichen nehmen die Bildelemente in zeitlicher Abtastrichtung einen hohen Amplitudenrang ein. Das Filter sucht dann den Medianwert in der stärker korrelierten örtlichen Nachbarschaft. Dies führt dazu, daß das Filter bei Bewegung automatisch auf örtliche Filterung umschaltet. Eine Bewegungsverschleifung, wie sie bei rekursiven Filtern zu beobachten ist, kann vom Medianfilter daher nicht hervorgerufen werden.

4 PALplus

Neben den neuen Fernsehsystemen wie D2-MAC und HDTV, die wegen der Belegung aller VHF/UHF-Kanäle nicht terrestrisch übertragen werden können, soll zukünftig auch das PAL-Verfahren auf das neue Bildseitenverhältnis 16:9 umgestellt werden. Im Gegensatz zum 4:3 Bildseitenformat, bietet das 16:9 Format für den Zuschauer einen verstärkten Eindruck der „Telepräsenz“, da dieses Bildseitenformat eher dem menschlichen Sehfeld entspricht. Eine wichtige Forderung für die erweiterte Fernsehnorm, die gleichzeitig auch die Grenzen des technisch Machbaren aufzeigt, ist die vollständige Kompatibilität zu den heutigen Empfängern.

4.1 Das Letterbox-Format

Das zu PAL kompatible neue Übertragungsverfahren PALplus (EDTV = Enhanced Definition Television) bedient sich zur Breitbildübertragung der sogenannten Letterbox-Codierung [5]. Das im Studio erzeugte 16:9 Bild wird dazu einer vertikalen Dezimationsfilterung unterworfen. Dabei wird das aus 576 aktiven Zeilen bestehende Bild auf 432 aktive Zeilen reduziert. Dieses aus 432 Zeilen bestehende Bildfenster wird auf einem herkömmlichen 4:3 Empfänger mit der korrekten Bildgeometrie dargestellt. Die aus den 576 Zeilen herausgefilterte Information wird nur von den neuen PALplus-Empfängern zur Rekonstruktion des vollen 16:9 Bildes benötigt. Damit diese Zusatzinformation nicht die bestehenden Empfänger stören kann, wird sie moduliert in Form von schwarzen Streifen zu je 72 aktiven Zeilen am oberen und unteren Bildrand, übertragen. Diese schon durch Kinofilmausstrahlungen bekannten Letterbox-Balken werden von den meisten Betrachtern nicht als störend empfunden. Der wesentliche Vorteil des Verfahrens liegt in der Übertragung der gesamten Bildinformation, auch für die bestehenden 4:3 Empfänger.

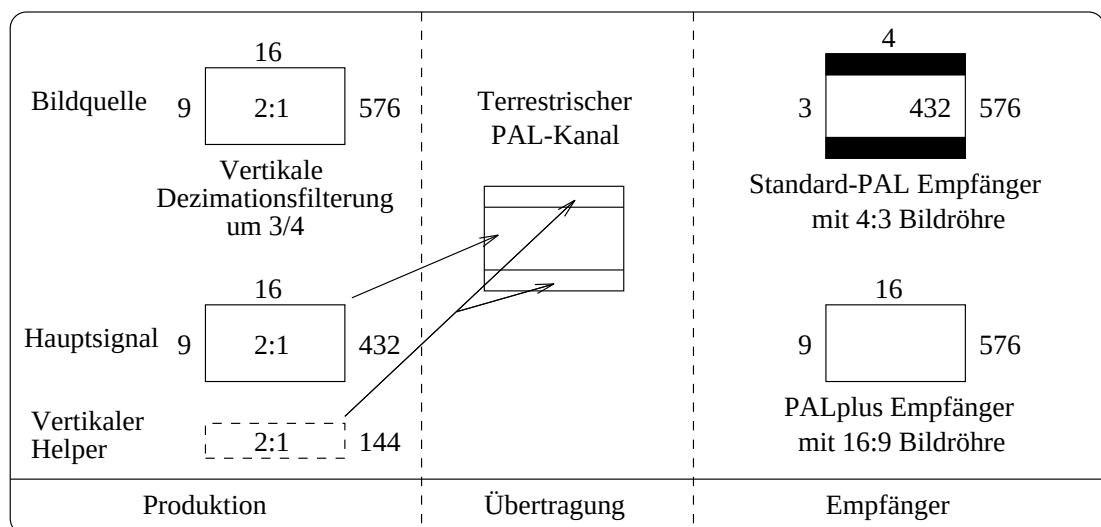


Abbildung 20: Prinzip der Letterbox-Signalverarbeitung

4.2 COLORplus

Ein weiterer Aspekt von PALplus ist die verbesserte Codierung und Decodierung von Chrominanz und Luminanz. Zwar läßt sich schon mit auf den Empfänger begrenzten Maßnahmen (IDTV) eine beträchtliche Verbesserung der Bildqualität erzielen, doch sind diese meist nur mit beträchtlichem Hardwareaufwand zu realisieren. Für die PALplus-Übertragung wird eine modifizierte PAL-Codierung verwendet, die den Namen COLORplus [19] trägt und die vollständig kompatibel zum PAL-Verfahren ist. Da diese COLORplus-Signale mit Standard-PAL-Signalen mischbar sind, kann die Umstellung auf COLORplus durch schrittweises Austauschen der Coder in den Fernsehstudios geschehen.

Das COLORplus-Konzept macht sich zunutze, daß der Farbträger bei den Fernsehstandards PAL, SECAM und NTSC zwischen Zeilen des ersten Halbbildes und bestimmten Zeilen des zweiten Halbbildes eine Phasendrehung um 180° aufweist.

Die Farbträgerfrequenz beim PAL-Verfahren ist in Gleichung 3 definiert. Wenn die Phase des Farbträgers im ersten Halbbild an einem bestimmten Ort x der Zeile n_1 gleich φ_1 ist, so ist sie im zweiten Halbbild am selben Ort x der Zeile $n_2 = n_1 + 312$ gleich φ_2 :

$$\begin{aligned}\varphi_2 &= \varphi_1 + 2\pi \cdot f_{sc} \cdot 312 \cdot t_h \\ &= \varphi_1 + 2\pi \cdot 312 \cdot \frac{f_{sc}}{f_h} \\ &\approx \varphi_1 + \pi + 2\pi \cdot 88530\end{aligned}\tag{8}$$

Die Phasenlage des Farbträgers benachbarter Zeilen ist in Abbildung 4.2 dargestellt.

Zeilennummern		U / V -Komponente
Halbbild 1	Halbbild 2	
	----- 313	180° / -90°
1	----- 314	0° / 90°
2	----- 315	-90° / 180°
3	----- 316	0° / 90°
4	----- 317	180° / -90°
5	----- 625	-90° / 180°
	----- 313	0° / 90°
	----- 625	180° / -90°
313	-----	0° / 90°

Abbildung 21: Phasenlagen des Farbträgers beim PAL-Verfahren

Wie schon in Kapitel 3.1 erläutert, vermischt sich beim PAL-Verfahren die hochfrequente Luminanz ab etwa 3 MHz mit der Chrominanz. Dies führt zu den bekannten Cross-Color und Cross-Luminanz Störungen. Aus Abbildung 4.2 geht hervor, daß

benachbarte Zeilen des ersten und des zweiten Halbbildes eine 180° Phasendrehung gegeneinander haben. Diese Tatsache lässt sich nun, unter bestimmten Voraussetzungen zur Trennung von Luminanz und Chrominanz nutzen. Wenn die hochfrequente Luminanz und die Chrominanz in den jeweiligen Zeilen benachbarter Halbbilder, die zum gleichen Vollbild gehören, gleich sind, so kann durch eine Addition beziehungsweise Subtraktion eine Trennung der Komponenten erfolgen. Die Gleichheit lässt sich dadurch erreichen, daß im Coder die U/V-Komponenten und die hochfrequente Luminanz der beiden Halbbilder in sogenannten Halbbildaddierern (Abbildung 4.2) jeweils gemittelt werden.

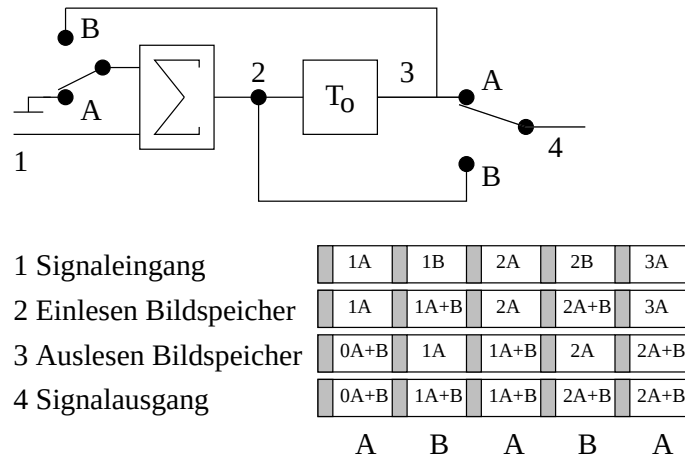


Abbildung 22: Digitaler Halbbildaddierer mit Halbbildspeicher

Diese gemittelten Halbbildkomponenten werden dann im ersten und im zweiten Halbbild übertragen. Nachfolgend wird das aus tieffrequenter Luminanz Y_t , hochfrequenter Luminanz Y_h und den modulierten Farbdifferenzsignalen zusammengesetzte Komponentensignal an dem Ort x einer Zeile n_1 bzw. n_2 betrachtet.

$$\begin{aligned}
 CVBS(x, n_2) &= Y_t(x, n_2) + Y_h(x, n_2) \\
 &+ U(x, n_2) \cdot \sin(\varphi_1 + \pi) + V(x, n_2) \cdot \cos(\varphi_1 + \pi) \cdot (-1)^{n_2}
 \end{aligned} \tag{9}$$

Mit $n_2 = n_1 + 312$ folgt daraus:

$$\begin{aligned}
 (-1)^{n_1} &= (-1)^{n_2} \\
 \sin(\varphi_1 + \pi) &= -\sin(\varphi_1) \\
 \cos(\varphi_1 + \pi) &= -\cos(\varphi_1)
 \end{aligned} \tag{10}$$

Die Halbbildaddierer für die hochfrequente Luminanz Y_h und die U/V-Komponenten bewirken:

$$Y_h(x, n_2) = Y_h(x, n_1) \iff \frac{Y_h(x, n_1) + Y_h(x, n_2)}{2} \tag{11}$$

$$\begin{aligned}
U(x, n_2) = U(x, n_1) &\Leftarrow \frac{U(x, n_1) + U(x, n_2)}{2} \\
V(x, n_2) = V(x, n_1) &\Leftarrow \frac{V(x, n_1) + V(x, n_2)}{2}
\end{aligned}$$

Damit wird aus Gleichung 9:

$$\begin{aligned}
CVBS(x, n_2) &= Y_t(x, n_2) + Y_h(x, n_1) \\
&- U(x, n_1) \cdot \sin(\varphi_1) - V(x, n_1) \cdot \cos(\varphi_1) \cdot (-1)^{n_1}
\end{aligned} \tag{12}$$

Durch eine spezielle Filterung im Coder und im Decoder können die Signalanteile Y_t und Y_h getrennt verarbeitet werden. Y_t wird konventionell übertragen, während die restlichen Komponenten im Coder den Halbbildaddierer durchlaufen müssen.

Für die Decodierung läßt sich nun folgende Betrachtung des hochfrequenten Komponentensignals anstellen:

$$CVBS_h(x, n_1) + CVBS_h(x, n_2) = 2 \cdot Y_h(x, n_1) \tag{13}$$

$$\begin{aligned}
CVBS_h(x, n_1) - CVBS_h(x, n_2) &= 2 \cdot U(x, n_1) \cdot \sin(\varphi_1) \\
&+ 2 \cdot V(x, n_1) \cdot \cos(\varphi_1) \cdot (-1)^{n_1}
\end{aligned} \tag{14}$$

Aus Gleichung 13 geht hervor, daß eine Halbbildaddition im Decoder die hochfrequente Luminanz, frei von Cross-Color, liefert. Gleichung 14 hingegen zeigt analog, daß eine Halbbildsubtraktion die Chrominanz, frei von Cross-Luminanz, erzeugt.

Eine Alternative hierzu besteht darin, die U/V-Komponenten nach der Demodulation zu behandeln. Der im Decoder regenerierte Farbträger hat an der Stelle x, n_1 die Phase φ_1 , und an der Stelle x, n_2 die Phase $\varphi_2 = \varphi_1 + \pi$. Nach der Demodulation in den Synchrondemodulatoren gilt dann für die U-Komponente:

$$\begin{aligned}
U_{dem}(x, n_1) &= (Y_h(x, n_1) + U(x, n_1) \cdot \sin(\varphi_1) \\
&+ V(x, n_1) \cdot \cos(\varphi_1) \cdot (-1)^{n_1}) \cdot \sin(\varphi_1) |_{TP}
\end{aligned} \tag{15}$$

Dabei gilt folgender Zusammenhang, bei entsprechender Wahl der Grenzfrequenz des Tiefpasses:

$$\sin(\alpha) \cdot \sin(\alpha) = \frac{1}{2} \cdot (1 - \cos(2\alpha)) |_{TP} = \frac{1}{2} \tag{16}$$

$$\cos(\alpha) \cdot \cos(\alpha) = \frac{1}{2} \cdot (1 + \cos(2\alpha)) |_{TP} = \frac{1}{2} \tag{17}$$

$$\sin(\alpha) \cdot \cos(\alpha) = \frac{1}{2} \cdot (0 + \sin(2\alpha)) |_{TP} = 0 \tag{18}$$

Daraus folgt dann:

$$U_{dem}(x, n_1) = Y_h(x, n_1) \cdot \sin(\varphi_1) |_{TP} + \frac{U(x, n_1)}{2} \quad (19)$$

$$\begin{aligned} U_{dem}(x, n_2) &= (Y_h(x, n_1) - U(x, n_1) \cdot \sin(\varphi_1) \\ &- V(x, n_1) \cdot \cos(\varphi_1) \cdot (-1)^{n_1} \cdot (-\sin(\varphi_1))) |_{TP} \end{aligned} \quad (20)$$

$$U_{dem}(x, n_2) = -Y_h(x, n_1) \cdot \sin(\varphi_1) |_{TP} + \frac{U(x, n_1)}{2} \quad (21)$$

$$U_{dem}(x, n_1) + U_{dem}(x, n_2) = U(x, n_1) \quad (22)$$

Analog dazu gilt für die V-Komponente:

$$\begin{aligned} V_{dem}(x, n_1) &= (Y_h(x, n_1) + U(x, n_1) \cdot \sin(\varphi_1) \\ &+ V(x, n_1) \cdot \cos(\varphi_1) \cdot (-1)^{n_1} \cdot \cos(\varphi_1)) |_{TP} \end{aligned} \quad (23)$$

$$V_{dem}(x, n_1) = Y_h(x, n_1) \cdot \cos(\varphi_1) |_{TP} + \frac{V(x, n_1)}{2} \quad (24)$$

$$\begin{aligned} V_{dem}(x, n_2) &= (Y_h(x, n_1) - U(x, n_1) \cdot \sin(\varphi_1) \\ &- V(x, n_1) \cdot \cos(\varphi_1) \cdot (-1)^{n_1} \cdot (-\cos(\varphi_1))) |_{TP} \end{aligned} \quad (25)$$

$$V_{dem}(x, n_2) = -Y_h(x, n_1) \cdot \cos(\varphi_1) |_{TP} + \frac{V(x, n_1)}{2} \quad (26)$$

$$V_{dem}(x, n_1) + V_{dem}(x, n_2) = V(x, n_1) \quad (27)$$

Der Subtraktionsprozeß vor der Demodulation (Gleichung 14) kann also durch eine Addition nach der Demodulation ersetzt werden. Diese Tatsache ermöglicht es alle Verarbeitungsschritte an den YUV-Komponenten zu vollziehen.

Da die Zeilen des ersten und zweiten Halbbildes die gleiche Information für die hochfrequente Luminanz und die Chrominanz tragen müssen, besitzen diese Komponenten nur noch 25 Bewegungsphasen pro Sekunde. Die Halbbildaddition bewirkt also eine zeitliche Unterabtastung. Weiterhin verlieren diese Komponenten die Hälfte ihrer vertikalen Auflösung, da nur noch die Zeilen eines Halbbildes zur Informationsübertragung zur Verfügung stehen. Um durch die zeitliche und vertikale Unterabtastung keine Aliaseffekte zu generieren, müssen die in Abbildung 4.2 gezeigten Auflösungsgrenzen berücksichtigt werden.

Der Wert Δy entspricht dabei dem geometrischen Abstand zweier benachbarter Zeilen. Die Auflösungsgrenzen gelten für die Chrominanz und für die hochfrequente Luminanz ab etwa 3 MHz. Um diese Grenzen einzuhalten, muß genaugenommen eine Vor- und Nachfilterung auf der Sende- und der Empfängerseite durchgeführt werden.

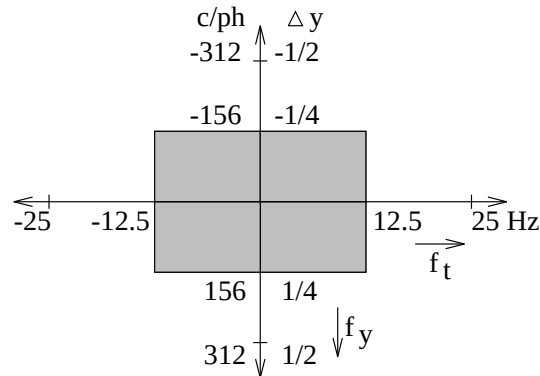


Abbildung 23: Verfügbarer Frequenzbereich bei COLORplus

Eine schwache räumlich-zeitliche Filterung wird schon durch die Halbbildaddition im Coder und im Decoder bewirkt. Zwar entspricht die Halbbildaddition nicht der systemtheoretisch korrekten Vor- und Nachfilterung, doch eine zusätzliche zeitliche Vorfilterung läßt keine sichtbaren Verbesserungen mehr erwarten [19]. In der Vertikalen führt die Begrenzung der Chrominanzbandbreite zu keiner sichtbaren Beeinträchtigung. Durch die Filterung werden lediglich bestehende Systemreserven ausgenutzt, da die Auflösung der Farbe auch in horizontaler Richtung (< 1.5 MHz) nur etwa ein Viertel der Luminanzauflösung (< 5 MHz) erreicht. Der vertikale und zeitliche Frequenzgang des Halbbildaddierers ist in Abbildung 4.2 dargestellt.

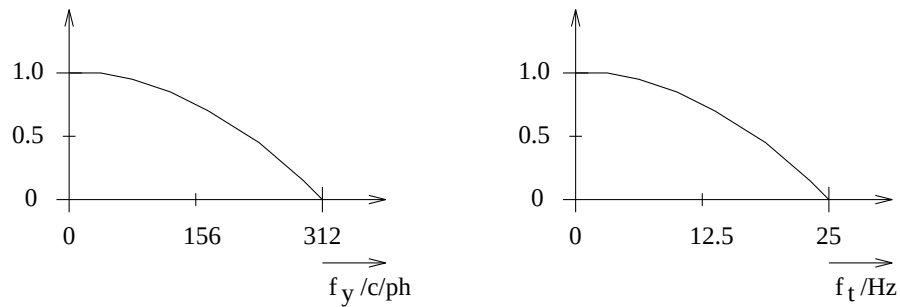


Abbildung 24: Frequenzgang des Halbbildaddierers

4.2.1 COLORplus-Coder

Abbildung 4.2.1 zeigt das Blockschaltbild eines COLORplus-Coders. Die RGB-Signale werden, wie beim PAL-Coder (Abbildung 2.1) auch, zuerst in Luminanzsignal (Y) und Chrominanzsignale (U/V) matriziert. Die tiefpaßgefilterten U/V -Komponenten und die hochfrequente Luminanz durchlaufen danach je einen Halbbildaddierer (Abbildung 4.2). Die Halbbildaddierer bewirken, daß diese Komponenten den gleichen Bildinhalt in den beiden Halbbildern eines Vollbildes haben. Die so modifizierten Farbdifferenzsignale werden dann in einem normalen PAL-Modulator auf den Farbträger aufmoduliert. Die resultierende Chrominanz wird mit der behandelten hochfrequenten Luminanz und der an die Laufzeit der Halbbildaddierer ($1/50 \text{ s} = 20 \text{ ms}$) angepaßten tieffrequenten Luminanz zum CVBS-Signal addiert.

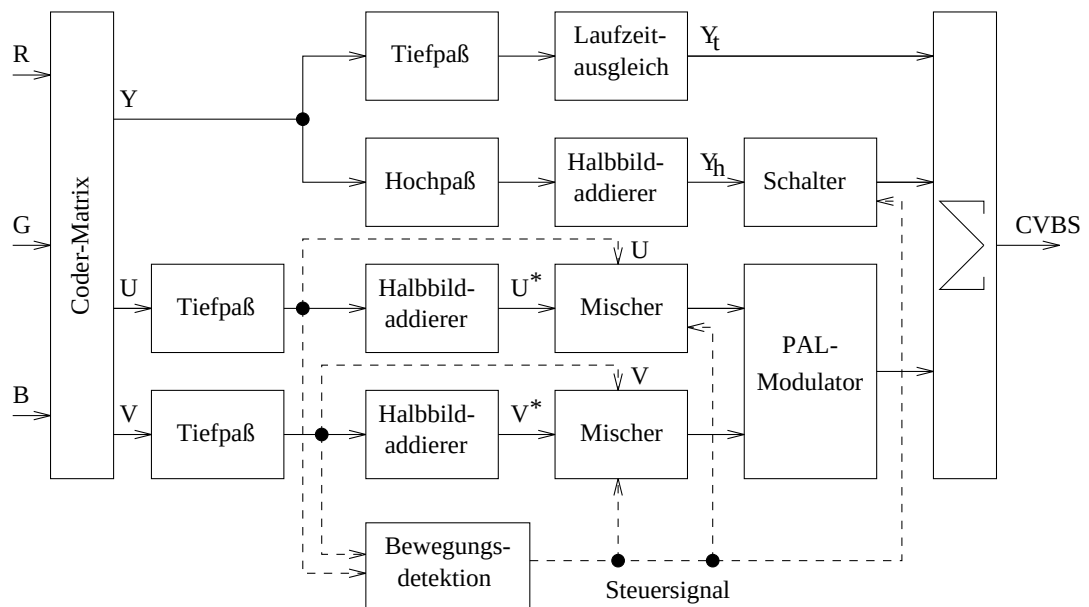


Abbildung 25: COLORplus-Coder mit Bewegungsdetektion

Das COLORplus-Verfahren eignet sich besonders gut für abgetastete Filmaufnahmen, bei denen die beiden Halbbilder eines Vollbildes aus derselben Bewegungsphase kommen und somit fast gleichen Bildinhalt aufweisen. Bei Bildquellen von elektronischen Kameras stammen die zwei Halbbilder eines Vollbildes aus zwei unterschiedlichen Bewegungsphasen. Durch die verminderte zeitliche Auflösung bei Farbsignalen entsteht beim COLORplus-Verfahren ein unerwünschtes Rucken an den Kanten schnell bewegter Farben mit hoher Sättigung. Dieses Problem kann mittels einer Bewegungsdetektion für solche kritischen Fälle gelöst werden. Zur Bewegungsdetektion wird eine geeignet gewichtete Differenz aus den U/V -Komponenten des ersten und zweiten Halbbildes berechnet. Nach der Betragsbildung und Gewichtung des Signals kann damit im Coder zwischen normalem COLORplus und einem sogenannten „Fall-Back“-Modus umgeschaltet werden. Dieser Modus erlaubt eine hohe zeitliche Auflösung für die Farbe, indem die Halbbildaddierer der U/V -Komponenten umgangen werden, und die Farbe in Standard-PAL übertragen wird.

Die hochfrequente Luminanz kann dabei komplett abgeschaltet werden, da sie in schnell bewegten Objekten vom Auge ohnehin nicht wahrgenommen werden kann.

4.2.2 COLORplus-Decoder

Das empfangene CVBS-Signal wird zur Luminanzrückgewinnung in einen hochfrequenten und einen tieffrequenten Anteil aufgespalten. Der Halbbildaddierer auf den hochfrequenten Anteil angewandt liefert die hochfrequente Luminanz (Gleichung 13). Die tieffrequente Luminanz wird laufzeitangepaßt. Zur Rückgewinnung der Chrominanz wird das CVBS-Signal hochpaßgefiltert und direkt in einem standardmäßigen PAL-Decoder demoduliert. Nach der Tiefpaßfilterung werden die U/V-Komponenten mit den Halbbildaddierern behandelt. Diese liefern dann nach Gleichung 22 und 27 die U/V-Komponenten, frei von Cross-Color durch hochfrequente Luminanzanteile.

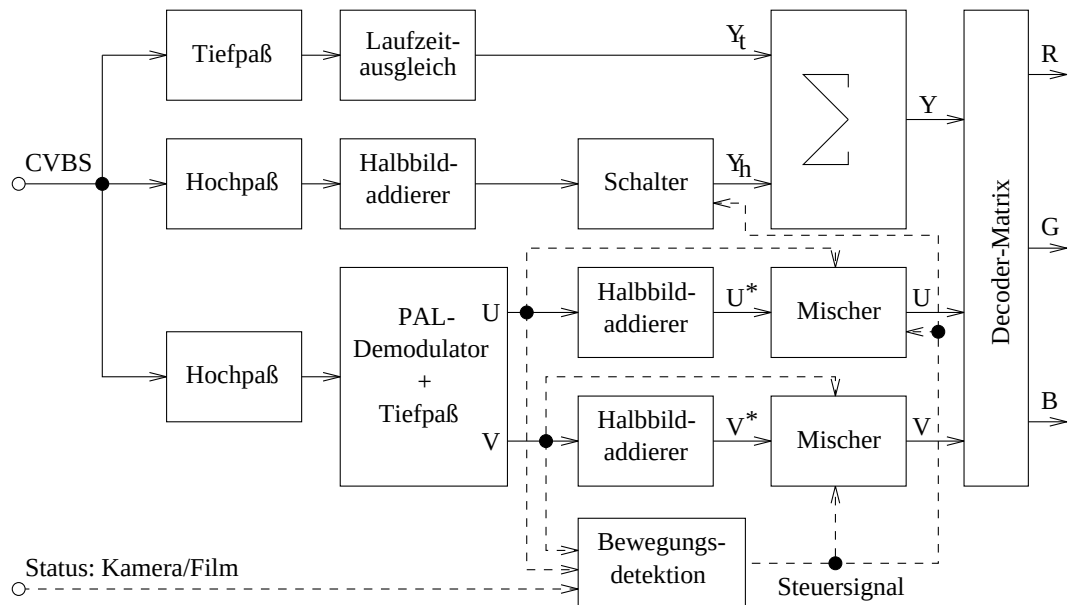


Abbildung 26: COLORplus-Decoder mit Bewegungsdetektion

Im Falle, daß die empfangene Sendung im Kamera-Modus übertragen wird, entsteht wiederum das Problem der schnell bewegten Farbkanten. Wie schon im Coder geschehen, ist dafür auch im Decoder eine Schaltung zur Bewegungsdetektion vorgesehen. Wenn also ein solcher kritischer Bildbereich detektiert wird, so werden im Decoder die Halbbildaddierer überbrückt und die hochfrequente Luminanz wird abgeschaltet. Wenn aber die zu decodierende Sendung ein abgetasteter Film ist, dann wird die Bewegungsdetektion im Decoder komplett abgeschaltet und es wird nur im COLORplus-Verfahren decodiert.

4.2.3 Bewegungsdetektion für COLORplus

Die Bewegungsdetektion im COLORplus-Coder und Decoder muß schnell bewegte Farbkanten mit hoher Farbsättigung erkennen, und ein Steuersignal bereitstellen,

welches den Codec zwischen dem COLORplus-Modus und dem „Fall-Back“-Modus umschalten kann. Definitionsgemäß muß die Bewegungsdetektion zwischen den beiden Halbbildern eines Vollbildes erfolgen. Als Eingangssignal für eine Schaltung zur Bewegungsdetektion werden demnach die demodulierten U/V-Komponenten eines Vollbildes benötigt. Getrennt nach U- und V-Komponente kann nun ein Differenzsignal zwischen benachbarten Punkten des ersten und des zweiten Halbbildes berechnet werden. Das Differenzsignal trägt Information über die Farbänderung zwischen den beiden Halbbildern. Simulationen mit einem solchen Verfahren haben jedoch gezeigt, daß sich damit keine sinnvolle Aussage über kritische Bildinhalte machen läßt. Die Gründe hierfür liegen vor allem an den beträchtlichen Cross-Color Störungen durch hochfrequente Luminanz, die als Chrominanz demoduliert wird. Dies führt dazu, daß in Bildbereichen mit feinen Luminanzstrukturen große Differenzsignale ermittelt werden, die ein unerwünschtes Umschalten auf den „Fall-Back“-Modus bewirken. Ein weiteres Problem bei der Bewegungsdetektion stellt das Zeilensprungverfahren (Interlace) dar. Da die zwei Halbbilder eines Vollbildes ineinander verkämmt sind, tragen benachbarte Punkte zweier Halbbilder unterschiedliche Farbinformationen. Es ist also bei einer Farbkante nicht möglich, sicher zu entscheiden, ob sie sich bewegt oder nicht.

Die genannten Probleme führen dazu, die Differenzbildung nicht auf zwei benachbarte Punkte zu beschränken, sondern das Bild in größere Bereiche zu unterteilen und zu untersuchen.

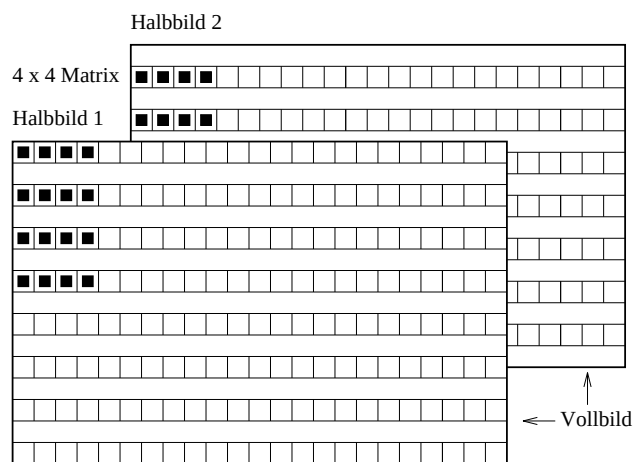


Abbildung 27: Matrix zur Bewegungsdetektion bei COLORplus

Es wird, wie in Abbildung 4.2.3 gezeigt, getrennt nach U- und V-Komponente, die Gesamtsumme der einzelnen Elemente einer Matrix berechnet. Die Matrizen-summen der beiden Halbbilder werden voneinander subtrahiert, und der Betrag gebildet (Abbildung 4.2.3). Die Signale von U- und V-Komponente werden anschließend addiert und auf einen Schwellwertentscheider gegeben. Wenn der Schwellwert S überschritten wird, so gibt die Schaltung ein Steuersignal ab, das für den gesamten Bereich, der von der Matrix überdeckt wird, im ersten und im zweiten Halbbild auf den „Fall-Back“-Modus umschaltet. Es wird also nicht weich zwischen COLORplus und dem „Fall-Back“-Modus übergeblendet, sondern hart umgeschaltet.

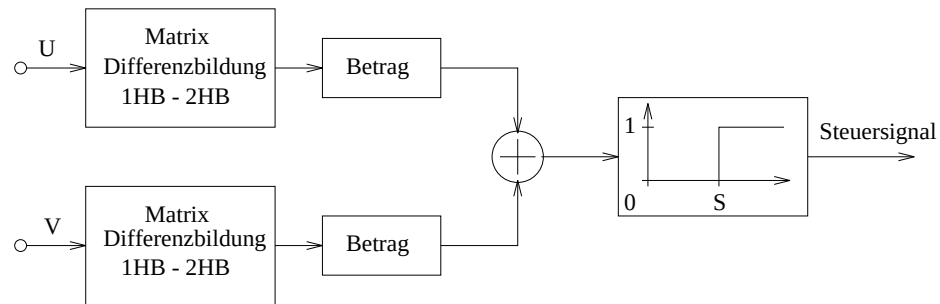


Abbildung 28: Schaltung zur Bewegungsdetektion bei COLORplus

Die Matrizengröße ist von entscheidendem Einfluß für die Funktion des Verfahrens. Zu ihrer Optimierung wurde ein im Kamera-Modus von COLORplus codiertes Testbild verwendet, das bewegte Farbbalken inmitten von horizontal hochfrequenten Luminanzstrukturen zeigt. Dabei stellte sich heraus, daß sich sowohl mit einer 5×5 -Matrix als auch mit einer 6×6 -Matrix die Cross-Color Störungen soweit herausmitteln, daß eine eindeutige Entscheidung zur Bewegungsdetektion in den meisten Fällen möglich ist. Größere Matrizen als die genannten tendieren dazu, auch die von der Bewegung selbst stammenden Signale zu mitteln. Bei kleineren Matrizen werden zwar bewegte Farbkanten erkannt, jedoch führen Cross-Color Störungen ebenfalls zu so großen Pegeln, daß keine Schwelle S mehr zur Unterscheidung bestimmbar ist.

4.2.4 Beispiele zur Bewegungsdetektion

Das Testbild in Abbildung 4.2.4 enthält im unteren Bildbereich, sich horizontal schnell bewegende Farbbalken. Die Farbbalken werden von horizontal hochfrequenten Luminanzmustern umgeben, die bei einer Decodierung nach Standard-PAL starke Cross-Color Störungen aufweisen.

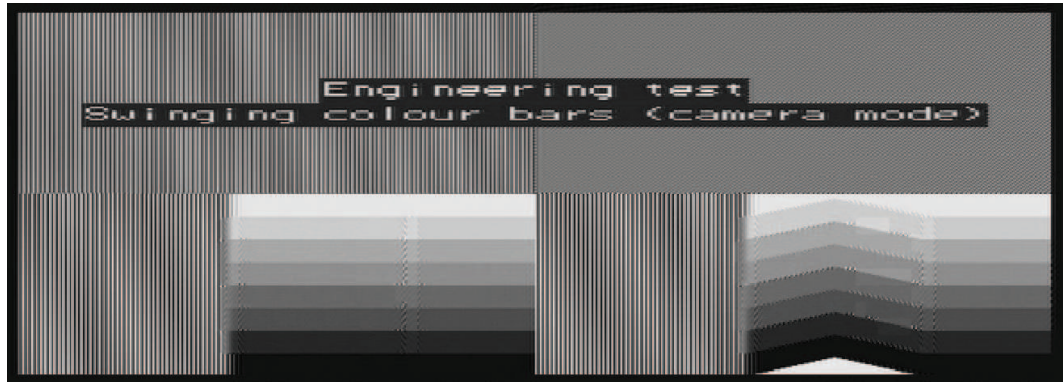


Abbildung 29: Testbild für die Bewegungsdetektion in COLORplus

Ziel der Bewegungsdetektion ist es dabei, die Randbereiche der bewegten Farbbalken als Bewegung zu erkennen, um auf den „Fall-Back“-Modus umschalten zu können. Andererseits muß der Decoder bei den auftretenden Cross-Color Störungen im COLORplus-Modus bleiben, da ansonsten die Störungen als farbige Bereiche zwischen dem Luminanzmuster deutlich sichtbar werden.

Die folgenden Abbildungen zeigen Bewegungssignale, für unterschiedliche Matrixdimensionen, vor dem Schwellwertentscheider. Je heller ein Bildbereich dargestellt wird, desto mehr Bewegung wurde in ihm erkannt. Umgekehrt gilt dann auch, daß in dunklen Bereichen nur wenig oder keine Bewegung erkannt wurde.



Abbildung 30: Bewegungssignal für eine Matrix der Dimension 2x2

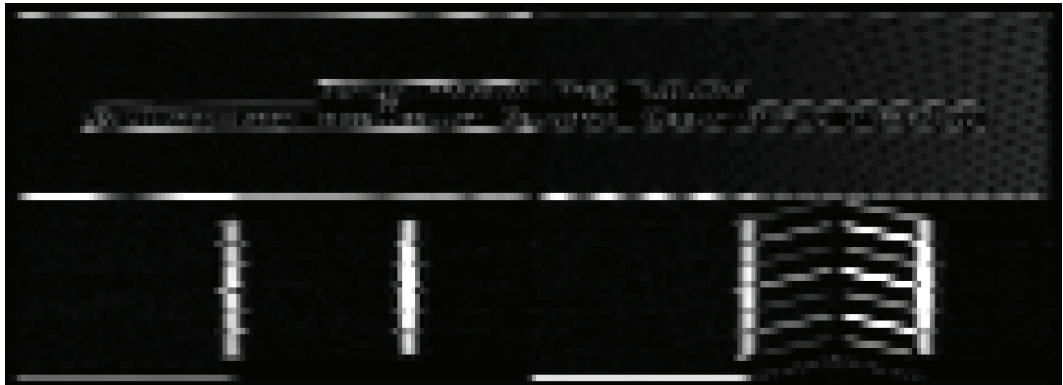


Abbildung 31: Bewegungssignal für eine Matrix der Dimension 4x4

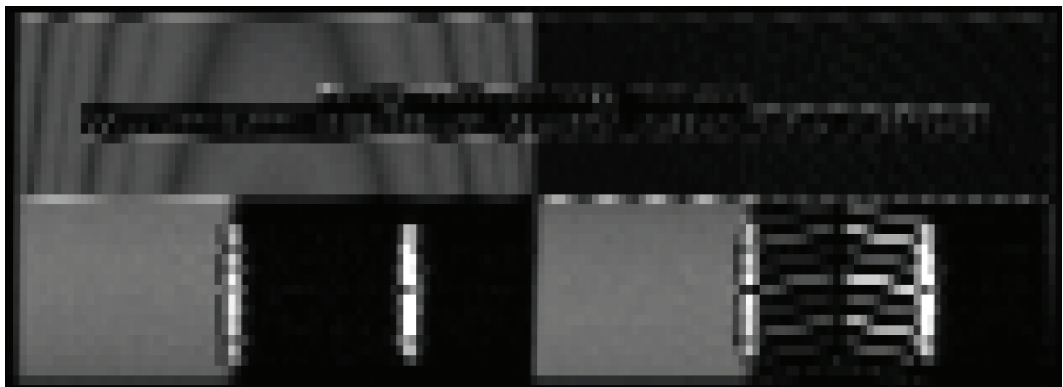


Abbildung 32: Bewegungssignal für eine Matrix der Dimension 6x6

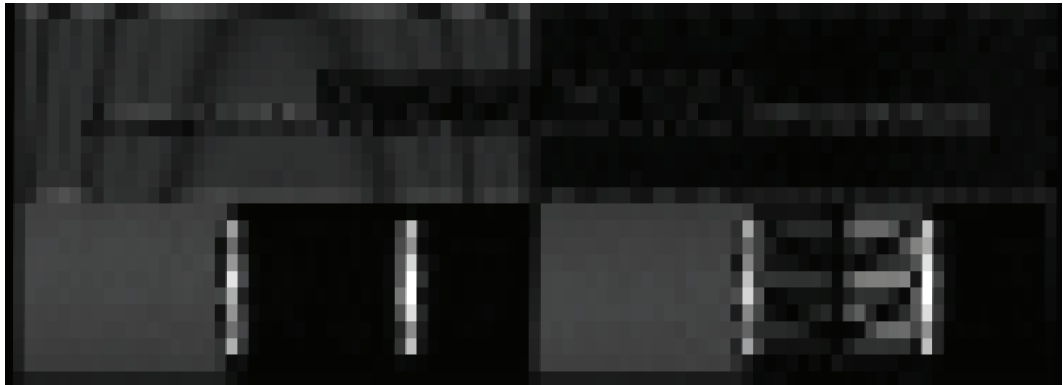


Abbildung 33: Bewegungssignal für eine Matrix der Dimension 10x10

Aus Abbildung 4.2.4 geht hervor, daß die Cross-Color Störungen bei einer Bewegungsdetektion mit einer 2×2 -Matrix noch so stark gewichtet werden, daß ein Schwellwert zur Umschaltung auf den „Fall-Back“-Modus nicht angegeben werden kann. Die 4×4 -Matrix aus Abbildung 4.2.4 mittelt zwar die Cross-Color Störungen in den Flächen der hochfrequenten Luminanz sehr gut heraus, doch bleiben an den Rändern der 4 Quadranten, wo ein Modulationssprung auftritt, noch immer sehr starke Pegel des Bewegungssignals. Im decodierten Bild erhält man deshalb subjektiv sehr störend wirkende farbige Punkte. Mit der 6×6 -Matrix aus Abbildung 4.2.4 lassen sich schon gute Gesamtergebnisse beobachten. Bei geeigneter Wahl des Schwellwerts kann hier gezielt an den bewegten Farbbalken auf den „Fall-Back“-Modus umgeschaltet werden. Noch besser schneidet im Vergleich die 10×10 -Matrix aus Abbildung 4.2.4 ab. Abbildung 4.2.4 zeigt das Bewegungssignal dieser Matrix nach dem Schwellwertentscheider. Die Tendenz zu noch größeren Matrizen ist jedoch nicht sinnvoll, da kleinere bewegte Objekte dann nicht mehr erkannt werden, und eine zu grobe Umschaltung zwischen COLORplus und „Fall-Back“-Modus erfolgt.



Abbildung 34: Bewegungssignal nach dem Schwellwertentscheider (10x10 Matrix)

4.3 PALplus-Codec

Abbildung 4.3 zeigt das Blockschaltbild des PALplus-Coders. Das Luminanzsignal wird von einem Quadratur-Spiegelfilter (engl. Quadrature-Mirror-Filter, QMF) in einen vertikalen Hoch- und einen Tiefpaßanteil aufgespalten. Ein Statussignal, das signalisiert ob im Kamera- oder Film-Modus codiert wird, steuert ob ein Intraframe- oder Intrafield-Quadratur-Spiegelfilter verwendet wird. Der Tiefpaßanteil wird einer vertikalen Dezimationsfilterung unterworfen, die die Anzahl der aktiven Zeilen von 576 auf 432 reduziert. Die Farbdifferenzsignale U/V werden, ebenso wie das tiefpaßgefilterte Luminanzsignal, von 576 auf 432 Zeilen dezimiert. Die 432 Zeilen der YUV-Komponenten gelangen dann zu einem COLORplus-Coder, wie in Abbildung 4.2.1 gezeigt, und werden dort zu einem CVBS-Signal codiert.

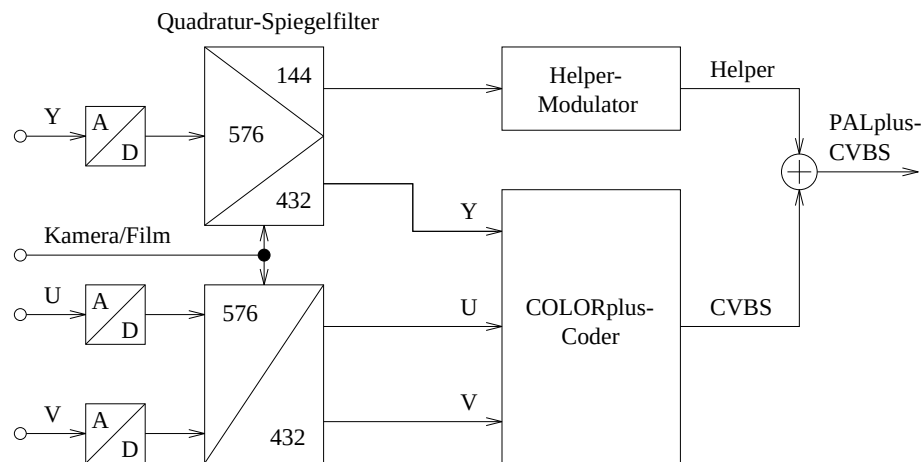


Abbildung 35: PALplus-Coder

Der Hochpaßanteil der Luminanz wird auf 144 Zeilen dezimiert und an den Helper-Modulator weitergegeben. Um das so entstandene Helper-Signal unempfindlich gegen Rauscheinflüsse zu machen, wird eine nichtlineare Amplituden-Vorverzerrung angewandt. Danach wird das Helper-Signal mit dem Farbträger moduliert, um die Energie des Signals um die Farbträgerfrequenz zu konzentrieren. Dadurch kann man das Helper-Signal gleichanteilsfrei übertragen, womit sich eine Störung der Synchronisation bei älteren kompatiblen Empfängern ausschließen läßt. Die Helper-Zeilen werden nach Abbildung 4.1 am oberen und am unteren Bildrand zu je 72 Zeilen aufgeteilt, und mit dem Ausgangssignal des COLORplus-Coders übertragen.

Das Blockschaltbild des PALplus-Decoders beginnt nach Abbildung 4.3 mit der A/D-Wandlung des CVBS-Signals. Das Hauptsignal (Abbildung 4.1) des PALplus-Signals wird im COLORplus-Decoder in die YUV-Komponenten decodiert. Dem COLORplus-Decoder schließt sich für die Farbdifferenzsignale U/V ein Interpolator an, der die 432 aktiven Zeilen des Hauptsignals auf die ursprünglichen 576 Zeilen interpoliert. Dieser Verarbeitungsschritt erfolgt in Abhängigkeit vom Modus des sendeseitigen Coders. Ist das empfangene Bild im Film-Modus codiert, so kann die Interpolation erfolgen, indem die Zeilen der beiden Halbbilder zuerst zum Vollbild verkämmt werden, und dann vertikal über alle Zeilen des Vollbildes interpoliert

wird. Sofern das empfangene Bild jedoch im Kamera-Modus codiert wurde, stammen die beiden Halbbilder aus unterschiedlichen Bewegungsphasen. Eine Interpolation über alle Zeilen des Vollbildes ist in diesem Falle nicht zulässig, da dies zu einer Bewegungsverschleifung führen würde [9]. Die Interpolation muß dann für beide Halbbilder getrennt erfolgen.

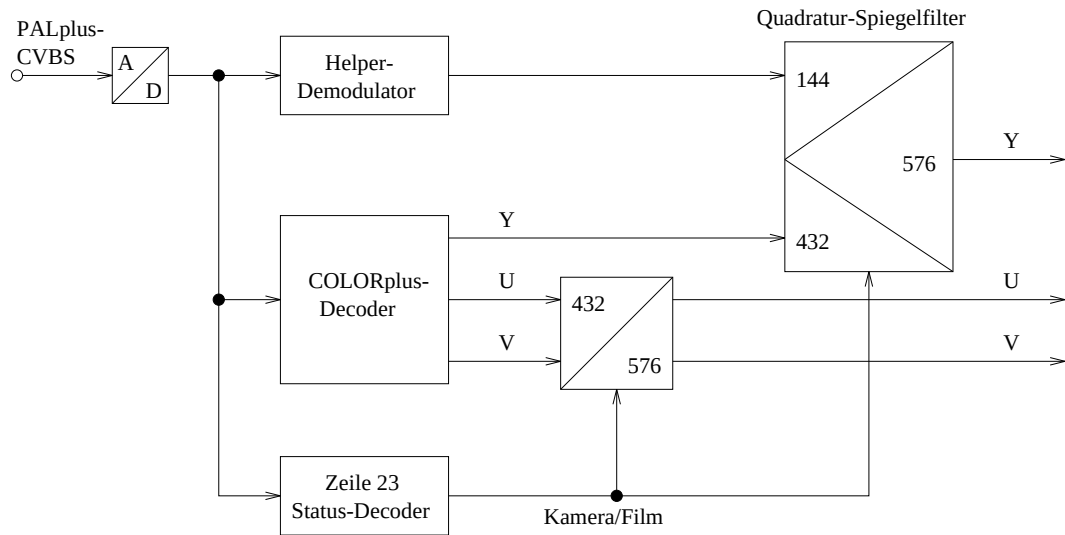


Abbildung 36: PALplus-Decoder

Zur Signalisierung über den aktuellen Betriebsmodus des PALplus-Coders werden, in der ersten Hälfte von Zeile 23, Signalbits übertragen, die darüber Auskunft geben. Im Status-Decoder werden diese Bits decodiert, und die entsprechenden Steuersignale an den Interpolator und das Quadratur-Spiegelfilter weitergegeben. Der Helper-Demodulator führt eine Synchrodemodulation des farbträgermodulierten Helper-Signals durch. Nach der Demodulation wird das Signal mit der zum Coder entgegengesetzten Kennlinie wieder entzerrt. Das 144 zeilige Helper-Signal, das die Information der vertikal hochfrequenten Luminanz trägt, und das 432 zeilige Luminanzsignal am Ausgang des COLORplus-Decoders werden dann in einem Quadratur-Spiegelfilter wieder zum Luminanzsignal zusammengesetzt.

4.4 Vertikale Bandaufspaltung

Das Prinzip der vertikalen Bandaufspaltung zur Letterbox-Codierung erfordert die Aufteilung des Luminanzsignals in einen Tiefpaßanteil und einen Hochpaßanteil. Der Tiefpaßanteil wird von 576 Zeilen auf 432 Zeilen dezimiert, während der Hochpaßanteil von ebenfalls 576 Zeilen auf nur 144 Zeilen dezimiert wird. Dies entspricht einer Bandaufspaltung im Verhältnis 3:4 (432/576), 1:4 (144/576). Sehr große Bedeutung kommt dabei den Filtern zu. Von ihnen wird gefordert, daß sie außerhalb des Durchlaßbereichs eine hohe Sperrdämpfung aufweisen. Nach der Dezimation überlappen sich sonst Nutzspektrum und Spiegelfrequenzen, was unweigerlich zu Aliasstörungen im kompatiblen Empfänger führt. Dahingegen entsteht bei zu niedrigen Grenzfrequenzen eine Bandlücke nach der Synthese im PALplus-Empfänger. Da es keine idealen Hoch- und Tiefpässe gibt, sind Aliasanteile in der Praxis nicht zu vermeiden. Diese Aliasstörungen lassen sich jedoch mit Quadratur-Spiegelfiltern im PALplus-Empfänger kompensieren.

4.4.1 Quadratur-Spiegelfilter (QMF)

Zur Bandaufspaltung mit nachfolgender Dezimation sind besonders die Quadratur-Spiegelfilter geeignet [9] [14]. Abbildung 4.4.1 zeigt das Blockschema einer solchen Filterbank, unterteilt in Coder- und Decoderseite. Sie führt eine Bandaufspaltung im Verhältnis 3:4, 1:4 durch.

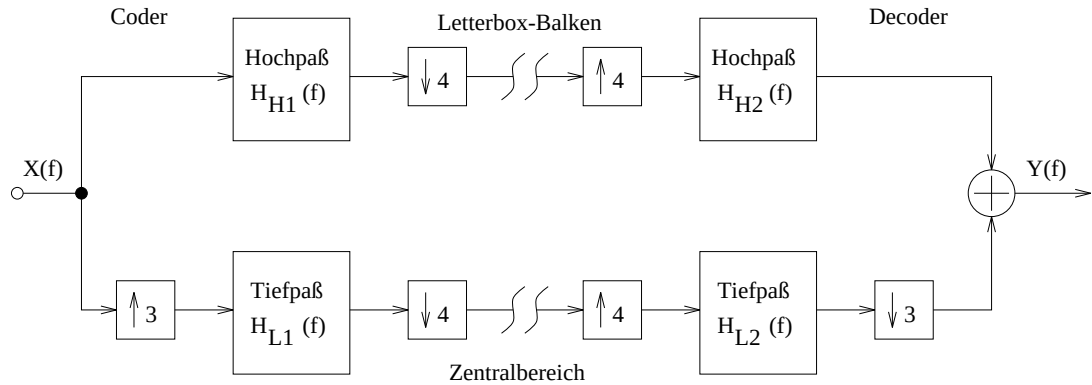


Abbildung 37: Filterbank zur Rasterkonversion

Die Quadratur-Spiegelfilter dienen zur Kompensation der Aliasstörungen, die bei der vertikalen Bandaufspaltung von den Überlappungsbereichen der Hoch- und Tiefpaßfilter hervorgerufen werden (Abbildung 4.4.1).

QMF erlauben eine nahezu perfekte Rekonstruktion von Amplitude und Phase der Originalsignale. Im Tiefpaßzweig des QMF erfolgt zunächst eine vertikale Interpolation um den Faktor 3. Daran schließt sich eine Tiefpaßfilterung und eine vertikale Dezimation um den Faktor 4 an. Diese wird bei der Bandsynthese im Decoder wieder rückgängig gemacht. Die dabei verwendeten Tiefpässe sind, abgesehen von einem Vorfaktor, gleich. Im Hochpaßzweig wird, nach der Hochpaßfilterung im Coder, um den Faktor 4 dezimiert. Im Decoder wird dies durch eine Interpolation

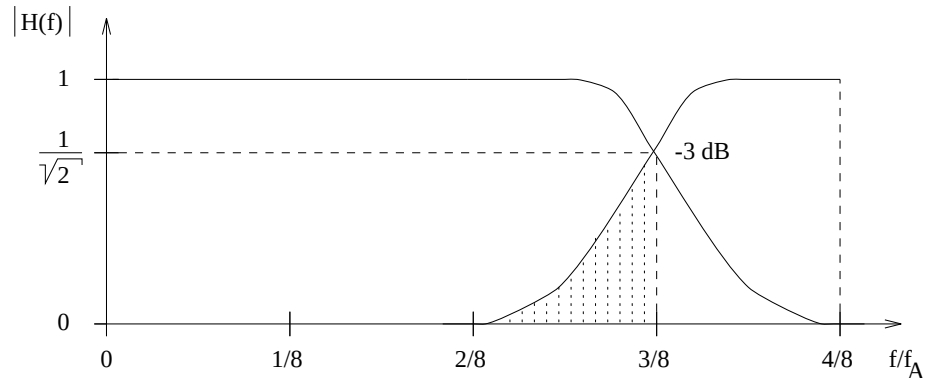


Abbildung 38: Hoch- und Tiefpaßfilter zur Bandaufspaltung 3:4, 1:4

um den Faktor 4 wieder rückgängig gemacht. Das folgende Hochpaßfilter hat einen um 180° gedrehten Phasengang zum Hochpaßfilter des Coders. Dadurch werden Aliasanteile im Ausgangssignal kompensiert. Abschließend werden die Signale des Tiefpaßzweigs und die des Hochpaßzweigs addiert.

Da der Frequenzgang des Ausgangssignals nach der Bandsynthese sich nicht von dem vor der Bandaufspaltung unterscheiden soll, müssen die Signale im Hoch- und Tiefpaßzweig, bei der Übergangsfrequenz, jeweils die halbe Signalamplitude haben. Das entspricht einer Dämpfung von 6 dB für die Filter bei der Übergangsfrequenz. Da sowohl im Hoch- als auch im Tiefpaßzweig je zwei Filter, eines im Coder und eines im Decoder, vorkommen, muß jedes Filter eine Dämpfung von 3 dB bei der Übergangsfrequenz aufweisen. Die Flanken der Filter im Hoch- und Tiefpaßzweig müssen außerdem spiegelsymmetrisch zur Übergangsfrequenz sein (Abbildung 4.4.1), wobei sich die quadrierten Anteile aus Hoch- und Tiefpaßzweig in der Summe zu Eins ergeben müssen [9].

4.4.2 Helper-Signalverarbeitung

Der Helper enthält den vertikalen Hochpaßanteil der Luminanz, der bei der Bandaufspaltung im QMF vom Luminanzsignal abgespalten wird. Um gleichanteilsfrei übertragen zu können, moduliert der vertikale Helper in Restseitenband- Amplitudenmodulation den Farbträger [14]. Es ergibt sich dabei eine nutzbare Bandbreite von etwa 4 MHz. Die Übertragung des Helpers erfolgt in den schwarzen Streifen des Letterbox-Bildes am oberen und unteren Bildrand. Zur Minimierung der Störeffekte auf kompatiblen PAL-Empfängern wird die Amplitude des Helpers auf ± 150 mV, symmetrisch zum Schwarzpegel begrenzt. Die Phasenlage des Farbträgers zur Modulation beträgt 0° , das Helper-Signal wird demnach von kompatiblen Empfängern als kaum störende dunkelblaue Chrominanz decodiert. In normalen Fernsehbildern ist dieser Effekt nicht wahrnehmbar, lediglich künstliche Testbilder mit diagonal sehr hochfrequenten Luminanzmustern bewirken deutlich sichtbare dunkelblaue Störmuster in den Letterbox-Balken.

Um die Rauschempfindlichkeit des Helper-Signals zu verringern, werden nicht-lineare Kompressionsmethoden verwendet. Subjektive Tests ergaben, daß vertikal

hochfrequente Luminanzsignale auf 83 % ihres Spitzenwerts „geclippt“ werden dürfen, ohne sichtbare Veränderungen hervorzurufen. Vertikal tieffrequente Bildausschnitte liefern keinen Beitrag zum Helper-Signal. Bei der Decodierung wird ihnen jedoch hochfrequentes Rauschen überlagert. Zur Unterdrückung dieser Störungen werden im Decoder alle Amplituden, die 9 % des Spitzenwerts nicht überschreiten, unterdrückt. Dieser Schwellwert wurde durch Simulationen mit einem Signal-Rauschabstand von 36 dB ermittelt [14]. Da das endgültige Aussehen der Coder- und Decoderkennlinie zur nichtlinearen Amplitudenkompression vom „PALplus Lenkungs-Ausschuß“ noch nicht bekanntgegeben wurde, sind in Abbildung 4.4.2 nur mögliche Beispiele für die Kennlinien zu sehen.

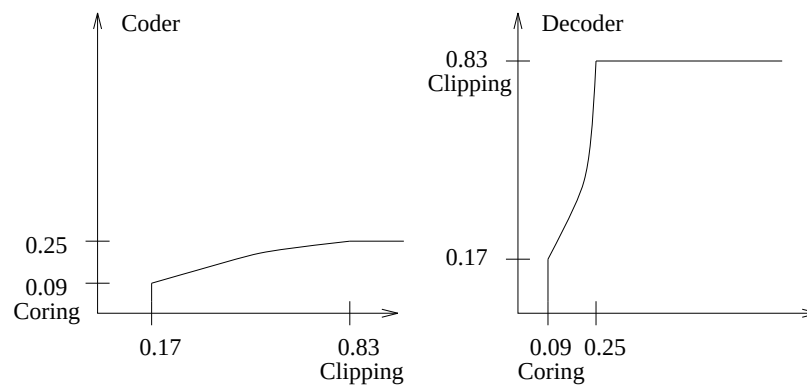


Abbildung 39: Nichtlineare Kennlinien im Coder und Decoder

4.5 Daten-Bits in Zeile 23

Um die Betriebsarten des PALplus-Decoders, abhängig vom aktuellen Modus des Coders, steuern zu können, werden Zusatzinformationen benötigt, die in der ersten Hälfte der Zeile 23 übertragen werden. Sie ist die erste aktive Zeile bei den Fernsehstandards 625/50/2:1, und noch nicht durch andere Informationsdienste wie Videotext und VPS belegt. Die Platzierung der Information in dieser Zeile erlaubt es, das Signal direkt an der Quelle mit den zugehörigen Daten zu versehen, und mit allen nachfolgenden Studiogeräten zu bearbeiten, ohne die Information zu verändern. Da die Zeile 23 im vertikalen Overscan der meisten Empfänger liegt, ist sie auf diesen Fernsehgeräten nicht sichtbar. Außerdem ist es möglich, die Information in Zeile 23 auch mit handelsüblichen Videogeräten aufzuzeichnen und wiederzugeben.

4.5.1 Codierung der Daten-Bits

Die codierte Information muß sehr resistent gegen Störungen sein, so daß eine sichere Decodierung auch bei Rauschabständen von weniger als 20 dB möglich ist. Die Information setzt sich aus dem „Wide-Screen-Letterbox“-Bit (WSL-Bit), 32 Daten-Bits und 8 Bits zur Fehlererkennung zusammen.

Die Daten-Bits werden in der ersten Hälfte von Zeile 23 übertragen. Die Taktfrequenz F_s beträgt 5 MHz, was einer Periodendauer T_s von 200 ns entspricht. Die

Codierung erfolgt mit einem Biphas-L Code, wobei ein Daten-Bit im NRZ-L Code mit je zwei Biphas-Elementen $T = 2 \cdot T_s = 400$ ns codiert wird (Abbildung 4.5.1).

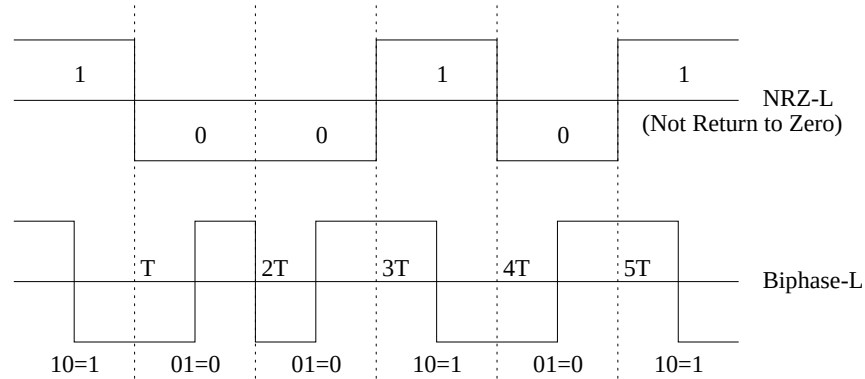


Abbildung 40: Biphas-L Codierung

Eine Ausnahme von der Biphas-L Codierung machen die Präambel und das WSL-Bit. Die Präambel ist eine Kennung, die vor den eigentlichen Daten-Bits übertragen wird, und ihren Anfang markiert. Das WSL-Bit kennzeichnet die im Letterbox-Format gesendeten Programme. Diese Information ist im Decoder so wichtig, daß sie besonders codiert wird, um auch bei den schlechtesten Empfangsverhältnissen sicher decodiert werden zu können.

Format der PALplus Daten-Bits								
Präambel	WSL-Bit	Byte 1	WSL-Bit	Byte 2	Byte 3	WSL-Bit	Byte 4	CRC
32 T_s	10 T_s	16 T_s	10 T_s	16 T_s	16 T_s	10 T_s	16 T_s	16 T_s

Beschreibung	Notation der Bits in Biphas-L Codierung	
Präambel	Hexadezimal: EAAA E946	
Anfangs-Kennung	Binär: 1110 1010 1010 1010 1110 1001 0100 0110	
Daten-Bit	0 = 01	1 = 10
WSL-Bit	0 = 0000011111	1 = 1111100000

Die Daten-Bytes 1 bis 4 und das CRC-Byte				
Daten-Byte 1	Daten-Byte 2	Daten-Byte 3	Daten-Byte 4	CRC-Byte
8 Bits b00 ... b07	8 Bits b08 ... b15	8 Bits b16 ... b23	8 Bits b24 ... b31	8 Bits b32 ... b39

4.5.2 Fehlererkennung mit dem CRC-8 Polynom

Zur sicheren Übertragung der PALplus Daten-Bits werden CRC-8 Check-Bits an die eigentlichen Daten-Bits angehängt. Diese CRC-8 Bits werden im Coder über die gesamten Bits der 4 Daten-Bytes berechnet. Hierzu werden die 4 Daten-Bytes als zusammenhängende Bitfolge (b00 ... 31) betrachtet, und durch das CRC-8 Polynom $x^8 + x^7 + x^5 + x^4 + x + 1$ geteilt. Der Rest dieser Operation umfasst 8 Stellen, und wird als CRC-8 Check-Bits an die Daten-Bits angehängt (b32 ... 39). Im Decoder wird die daraus resultierende Bitfolge (b00 ... b39) wiederum durch das CRC-8 Polynom

geteilt. Nur wenn kein Fehler bei der Übertragung und Decodierung aufgetreten ist, ergibt sich der Rest dieser Polynomdivision zu 0. Der Decodieralgorithmus ist auf dem Chip 9401 von Fairchild erhältlich [3]. Abbildung 4.5.2 zeigt eine Schaltung zur Polynomdivision mit dem CRC-8 Polynom, wie sie ebenfalls zur Fehlererkennung eingesetzt werden kann [26].

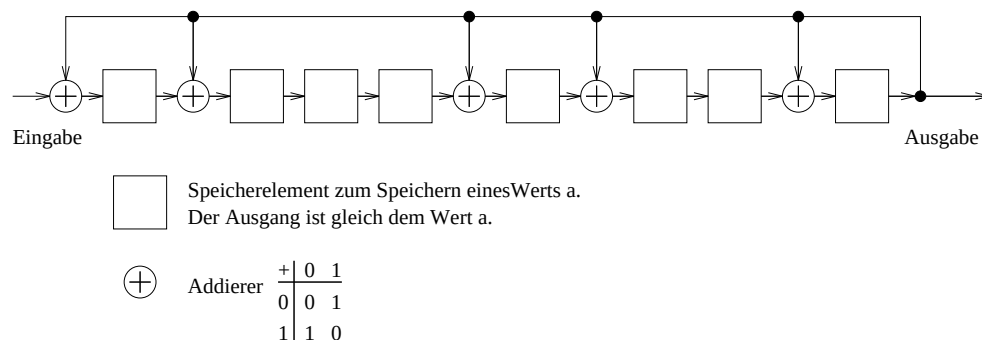


Abbildung 41: Schaltung zur Polynomdivision durch das CRC-8 Polynom

Am Anfang müssen alle Speicherstellen auf 0 gesetzt werden. Der Ausgangswert bleibt dann für die ersten 8 Verschiebungen gleich 0, bis das erste Bit des Eingangs den Ausgang erreicht. Nach einer Gesamtzahl von 40 Verschiebungen ist der volle Quotient am Ausgang erschienen. Der für die Fehlererkennung interessierende Rest steht im Schieberegister. Wenn nicht alle Speicherstellen des Schieberegisters gleich 0 sind, so hat die Schaltung einen oder mehrere Fehler erkannt. Der PALplus-Decoder darf in diesem Falle nicht auf die Daten-Bits reagieren. Da die Daten-Bits in jedem Vollbild (25/s) wiederholt werden, kann auf eine Fehlerkorrektur verzichtet werden.

4.5.3 Bedeutung der Daten-Bits

Die folgenden Tabellen geben Auskunft über die Bedeutung der einzelnen Daten-Bits. Wenn ein Daten-Bit noch keine Bedeutung hat, so ist es mit „nicht belegt“ gekennzeichnet.

1. Daten-Byte 1: Abbildungsverhältnis, Film-Bit und Untertitel

- b00 ... b02 Letterbox-Format und Position

Daten-Byte: 1 / Bits: b00 b01 b02			
WSL	b02 b01 b00	Abbildungsverhältnis	Zeilenzahl
0	000	4:3 Vollformat	576 Zeilen
1	001	14:9 Letterbox Mitte	504 Zeilen
1	010	14:9 Letterbox Oben	504 Zeilen
1	011	16:9 Letterbox Mitte	430 Zeilen
1	100	16:9 Letterbox Oben	430 Zeilen
1	101	>16:9 Letterbox Mitte	
0	110	nicht belegt	
0	111	16:9 Vollformat	576 Zeilen
nicht in der terrestrischen Ausstrahlung			

- b03 nicht belegt
- b04 Film-Bit

Daten-Byte: 1 / Bit: b04	
0	50 Hz Kamera-Modus
1	25 Hz Film-Modus

- b05 ... b07 Untertitel

Daten-Byte: 1 / Bits: b05 b06 b07		
b07 b06 b05	Position der Untertitel	Art der Untertitel
000	keine Untertitel	
001	im aktiven Bildbereich	keine Teletext Untertitel
010	im aktiven Bildbereich	Teletext Untertitel
011	nicht im aktiven Bildbereich	keine Teletext Untertitel
100	nicht im aktiven Bildbereich	Teletext Untertitel
101		nur Teletext Untertitel
110	nicht belegt	
111	nicht belegt	

2. Daten-Byte 2: Reserviert für neue Fernsehstandards

- b08 PALplus-Bit

Daten-Byte: 2 / Bit: b08	
0	Standard Service (PAL/SECAM)
1	PALplus

- b09 ... b15 nicht belegt

3. Daten-Byte 3: Reserviert für die Tonübertragung

- b16 ... b23 nicht belegt

4. Daten-Byte 4: Reserviert für den zukünftigen Gebrauch

- b24 ... b31 nicht belegt

5. Daten-Byte 5: Fehlererkennung

- b32 ... b39 CRC-8 Fehlererkennungs-Bits

4.5.4 Vorschriften zur Arbeitsweise des Decoders

Die bisher bestehenden Breitbild-Quellen haben unterschiedliche Abbildungsverhältnisse. Die gebräuchlichsten davon sind 1.33, 1.66, 1.78, 1.85 und 2.35. Der genaue Wert ist abhängig davon, ob es sich bei der Quelle um Breitbild-Filme, Cinemascope-Filme, HDTV-Produktionen oder andere Formate handelt. Die PALplus-Bits, in der Tabelle über Abbildungsverhältnis und Zeilenzahl, werden dann wie folgt gesetzt:

Abbildungsverhältnis				PALplus-Signalisierung
	a	<	1.46	4:3
1.46	<	a	< 1.66	14:9
1.66	<	a	< 1.90	16:9
1.90	<	a		> 16:9

Das 16:9 Letterbox-Format enthält 430 aktive Zeilen. Die Position dieser Zeilen im ersten und im zweiten Halbbild ist wie folgt:

16:9	Halbbild 1	Halbbild 2
Erste Zeile	60	372
Letzte Zeile	274	586

Das 14:9 Letterbox-Format enthält 504 aktive Zeilen. Die Position dieser Zeilen im ersten und im zweiten Halbbild ist wie folgt:

14:9	Halbbild 1	Halbbild 2
Erste Zeile	42	354
Letzte Zeile	293	605

5 Programmierung des PALplus-Decoders

Zur Simulation der bisher vorgestellten Blockschaltungen eignet sich besonders das KHOROS-System, das an der Universität von New Mexico entwickelt wurde [2]. Es handelt sich dabei um eine Software-Umgebung zur Entwicklung von Bild- und Signalverarbeitungs-Anwendungen. Durch zahlreiche Tools wird die Entwicklung einer neuen Anwendung auf die Programmierung des entsprechenden Algorithmus begrenzt. Das Design der neuen Programmoberfläche erfolgt interaktiv mit dem KHOROS-Programm „composer“. Zu dieser neuen Programmoberfläche erzeugt der „ghostwriter“ ein Programmgerüst in der Sprache „C“, das schon alle Routinen zur Parameterübergabe und ihrer Deklaration enthält. Die Aufgabe des Programmierers beschränkt sich dann weitgehend auf das Erstellen des restlichen Programmcodes zur Ausführung der gewünschten Aufgabe. Auch hier stehen zahlreiche Funktionen und Makros zur Verfügung, die diesen Schritt vereinfachen. Die Ein- und Ausgabeschnittstellen der entwickelten Programme können dann übersichtlich auf der graphischen Simulationsoberfläche „cantata“ miteinander verbunden werden. Zur Datenübergabe zwischen den einzelnen Programmen gibt es ein besonderes Datenformat, das KHOROS VIFF-Format (Visualization/Image File Format). Zusammen mit den schon bestehenden Grundfunktionen, kann der Anwender über „cantata“ auf alle installierten Routinen zur Signalverarbeitung zugreifen, sie zu komplexen Blockschaltungen zusammensetzen und ausführen.

5.1 Funktionsgruppen des Decoders

Nachfolgend werden die einzelnen Programmroutinen des PAL-Decoders beschrieben. Der COLORplus-Decoder, als Teil von PALplus, ist komplett in einem Programm zusammengefaßt. Zum PALplus-Decoder ist der Helper-Demodulator und der Status-Decoder für Zeile 23 verfügbar. Es handelt sich dabei um Routinen, die unter dem KHOROS-System eingesetzt werden können. Die Funktion jeder Routine, die benötigten Parameter und Dateiformate, werden an Hand der einzelnen Programmoberflächen (Graphical User Interface) erläutert.

5.1.1 raw2cvbs

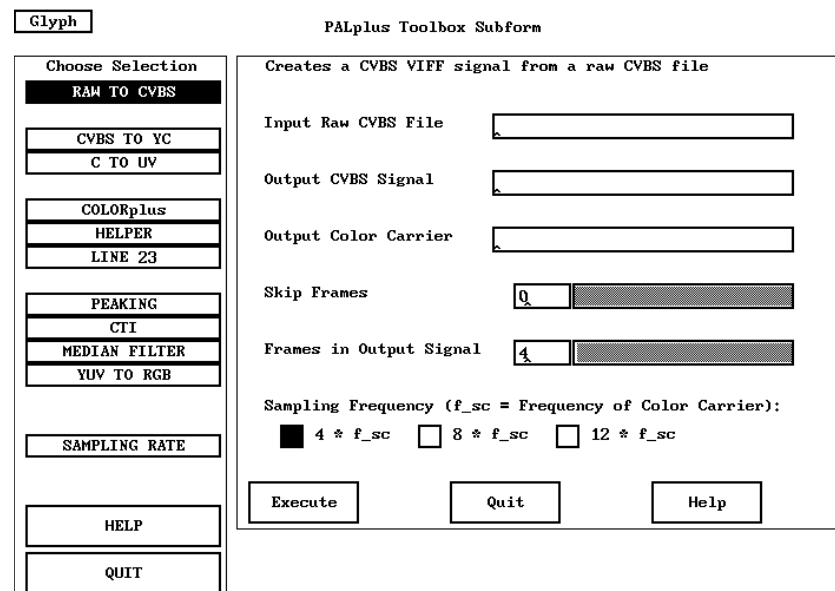


Abbildung 42: Programmoberfläche (GUI) von „raw2cvbs“

„raw2cvbs“ konvertiert ein „rohes“ CVBS-Signal, wie es zum Beispiel der D3 Recorder liefert, in eine VIFF CVBS-Datei. Die VIFF CVBS-Datei beginnt, nach der PAL B/G Norm, mit Halbbild 1 oder 3. Zusätzlich wird der Farbträger, zur Farbdemodulation, bereitgestellt.

- Eingabe:
 1. Rohes CVBS-Signal (BYTE), mit 8 Bit quantisiert. Das Signal muß mindestens ein Vollbild umfassen, und mit dem 4, 8 oder 12-fachen der Farbträgerfrequenz abgetastet sein. Der Schwarzpegel muß mindestens bei 16 liegen. Der Pegel der Synchronimpulse muß gleich 0 sein.
- Parameter:
 1. Anzahl der Vollbilder, die am Anfang übersprungen werden sollen.
 2. Anzahl der Vollbilder, die ausgegeben werden sollen.
 3. Abtastfrequenz des „rohen“ CVBS-Signals (4, 8 oder 12-fache Farbträgerfrequenz).
- Ausgabe:
 1. VIFF CVBS-Datei (BYTE 0 ... 255), die mit einer ungeraden Halbbildnummer beginnt. Der Pegel der Synchronimpulse liegt bei 0. Der Schwarzpegel beträgt 62 und der Pegel für Weiß beträgt 207. Der Maximalpegel für Weiß plus Chrominanz liegt bei 255.
 2. VIFF Farbträger-Datei (BYTE 128 ± 127) mit 0° relativer Phase.

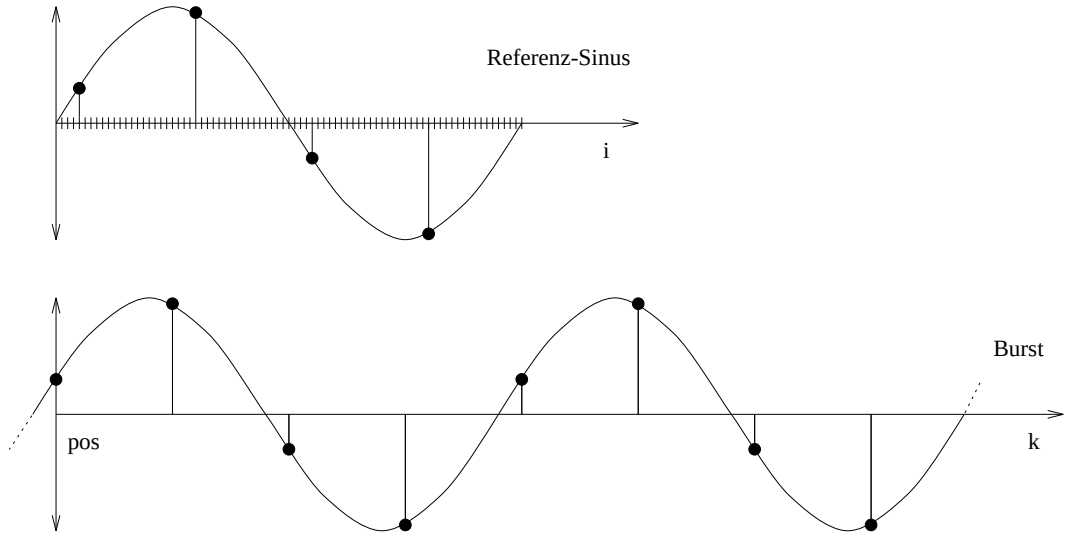


Abbildung 43: Ermitteln der Farbträgerphase

Die Farbträgerphase wird durch eine Abschätzung mit einem Referenz-Sinus bestimmt. Um einen Phasenfehler von 1° nicht zu überschreiten, sind beim Referenz-Sinus mehr als 360 diskrete Werte pro Periode 2π zu bestimmen. Der Phasenfehler beträgt dann $< \pm 1^\circ$. Im vorliegenden Algorithmus werden $4 \cdot 8 \cdot 12 = 384$ diskrete Werte pro Periode mit der Amplitude des Burstes berechnet, und im Feld `ref[]` gespeichert. Danach wird eine Periode des Burstes an der Stelle `pos` aus dem CVBS-Signal herausgetastet, und im Feld `fsc[]` gespeichert. Um den Burst mittelwertfrei zum Vergleich zu erhalten, wird der Schwarzwert abgezogen. Es soll nun die relative Phase des Burstes an der Stelle `pos` bestimmt werden. Die weiteren Betrachtungen erfolgen für die Abtastfrequenz $4 \cdot f_{sc}$.

$$err_i = \sum_{k=0}^{4-1} \left| fsc_k - ref_{i+k \cdot \frac{384}{4}} \right| \quad (28)$$

mit $i = 0 \dots 384 - 1$. Feld `err[]` enthält nun den relativen Fehler zwischen Referenz-Sinus und Burst an der Stelle `pos`, zu den Phasen $\beta = 2\pi \cdot i/384$. An der Stelle i mit dem kleinsten Fehler $err_i|_{min}$ wird dann die Burstphase β berechnet. Da der Burst zeilenweise um $\pm 135^\circ$ alterniert, ist die Burstphase β nicht gleich der Farbträgerphase α an der Stelle `pos`. In Halbbild 1 ist der erste Burst in Zeile 7 zu finden, in Halbbild 3 jedoch in Zeile 6 [15]. Dieses Kriterium ermöglicht die Unterscheidung zwischen Halbbild 1 und Halbbild 3. Die relativen Phasen der ersten Bursts jedes Halbbildes sind gleich $+135^\circ = 3\pi/4$. Die Farbträgerphase zu Beginn des CVBS-Signals ist dann:

$$\alpha = \beta - \frac{3\pi}{4} - \left(\frac{2\pi}{4} \cdot pos \right) \text{ modulo } (2\pi) \quad (29)$$

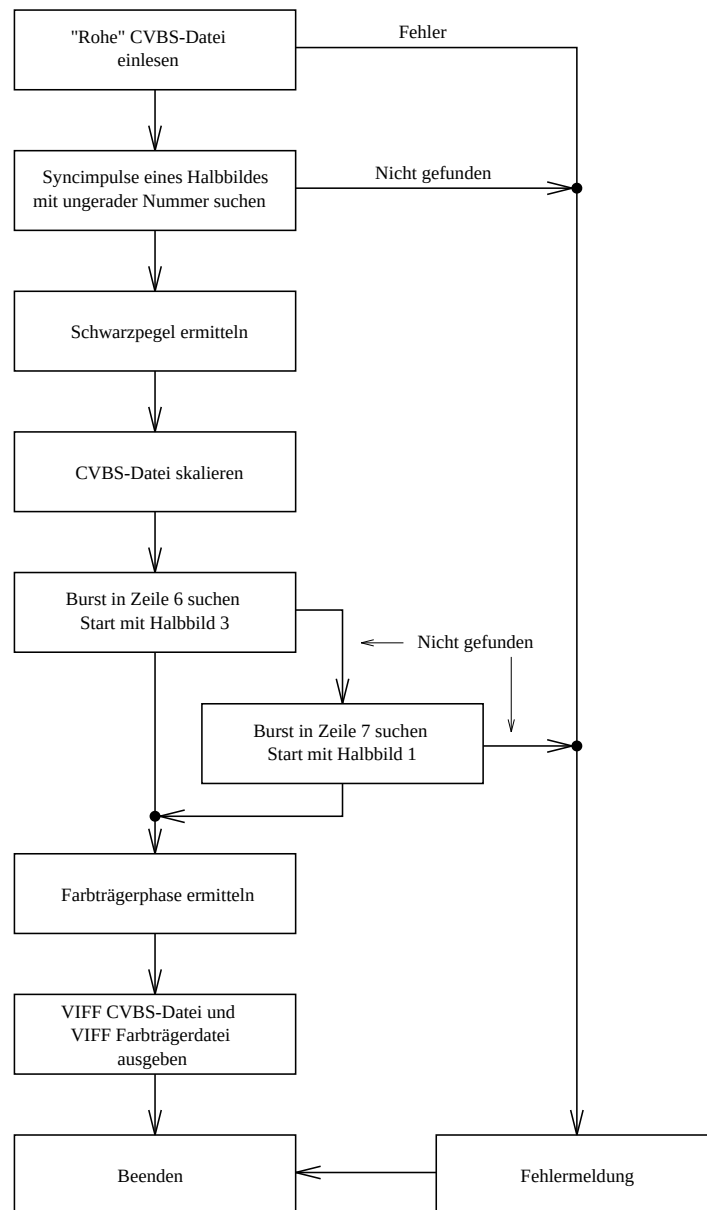


Abbildung 44: Flußdiagramm von „raw2cvbs“

5.1.2 cvbs2yc

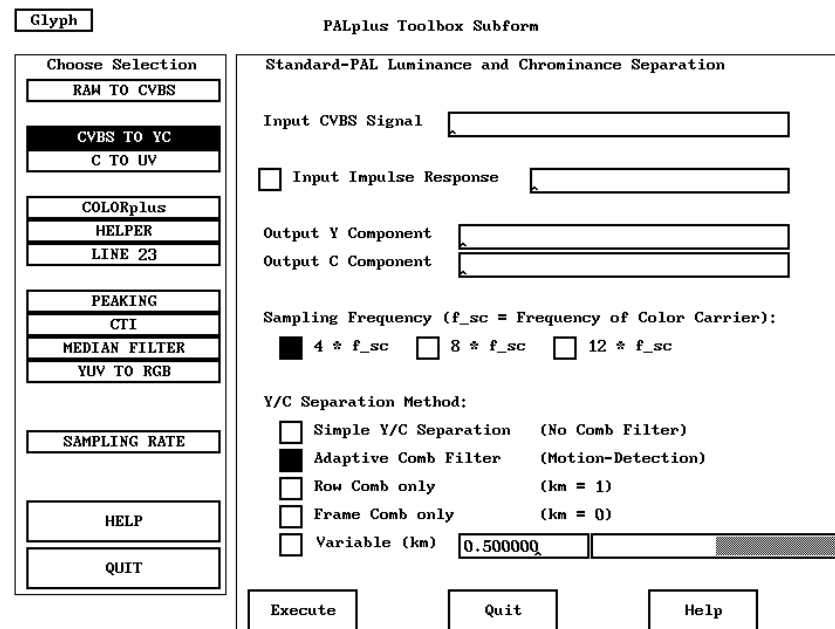


Abbildung 45: Programmoberfläche (GUI) von „cvbs2yc“

„cvbs2yc“ trennt Chrominanz und Luminanz.

- Eingabe:
 1. VIFF CVBS-Datei (BYTE 0 ... 255), wie sie „raw2cvbs“ liefert.
 2. Impulsantwort eines Bandpaßfilters (FLOAT). Die Anzahl der Koeffizienten des Filters muß ungeradzahlig sein. Der Bandpaß sollte symmetrisch zur Farbträgerfrequenz sein (Durchlaßbereich: $4.43 \text{ MHz} \pm \approx 1.3 \text{ MHz}$).
- Parameter:
 1. Abtastfrequenz der VIFF CVBS-Datei (4, 8 oder 12-fache Farbträgerfrequenz).
 2. Methode zur Chroma/Luma-Trennung:
 - (a) Einfache Y/C-Trennung mit Bandpaß $BP(f)$ zur Chrominanztrennung und Tiefpaß $1 - BP(f)$ zur Luminanztrennung.
 - (b) Adaptives Bild- und Zeilenkammfilter [34].
 - (c) Nur Zeilenkammfilter.
 - (d) Nur Bildkammfilter.
 - (e) Mischsignal von Bild- und Zeilenkammfilter.
- Ausgabe:
 1. VIFF Luminanz-Datei (BYTE 16 ... 235).
 2. VIFF Chrominanz-Datei (BYTE 128 ± 127).

5.1.3 c2uv

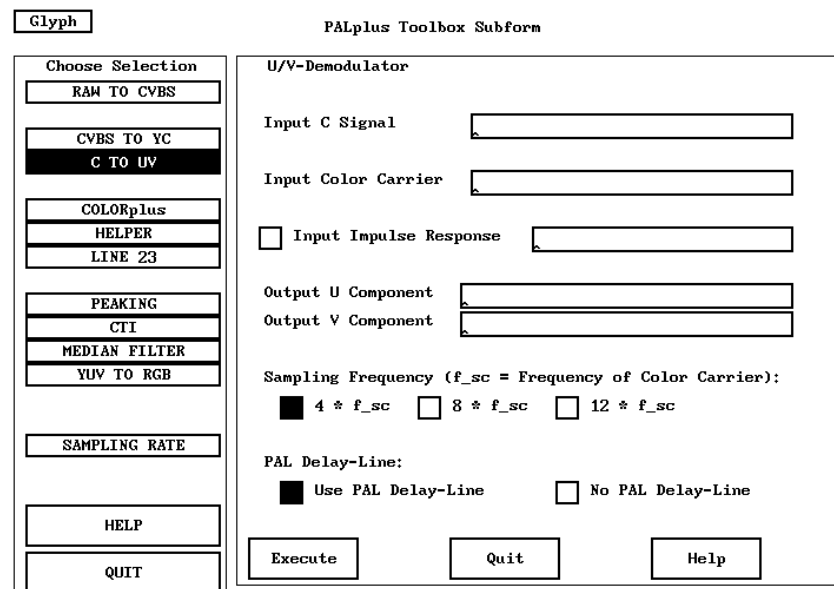


Abbildung 46: Programmoberfläche (GUI) von „c2uv“

„c2uv“ führt eine Synchrondemodulation der Chrominanz durch.

- Eingabe:

1. VIFF Chrominanz-Datei (BYTE 128 ± 127), wie sie „cvbs2yc“ liefert.
2. VIFF Farbträger-Datei (BYTE 128 ± 127), wie sie „raw2cvbs“ liefert.
3. Impulsantwort eines Tiefpaßfilters (FLOAT). Die Anzahl der Koeffizienten des Filters muß ungeradzahlig sein. Der Tiefpaß ist hinter den Demodulator geschaltet (Grenzfrequenz: ≈ 1.3 MHz).

- Parameter:

1. Abtastfrequenz der VIFF Chrominanz/Farbträger-Datei (4, 8 oder 12-fache Farbträgerfrequenz).
2. PAL Verzögerungsleitung ($64.057 \mu\text{s}$):
 - (a) Standard-PAL (Use PAL Delay-Line).
 - (b) Simple-PAL (No PAL Delay-Line).

- Ausgabe:

1. VIFF U-Datei (BYTE 128 ± 112).
2. VIFF V-Datei (BYTE 128 ± 112).

5.1.4 yuv2rgb

Abbildung 47: Programmoberfläche (GUI) von „yuv2rgb“

„yuv2rgb“ matrizen die YUV-Komponenten in RGB-Signale. Die YUV-Komponenten müssen nach CCIR-Rec.601 skaliert sein [24], und gleiche Dateilänge haben (Abtastfrequenz).

- Eingabe:

1. Y-Komponente (BYTE 16 ... 235). Der Schwarzpegel liegt bei 16, und der Weißpegel bei 235.
2. U-Komponente (BYTE 128 ± 112). Keine U-Komponente für den Wert 128.
3. V-Komponente (BYTE 128 ± 112). Keine V-Komponente für den Wert 128.

- Ausgabe:

1. R-Komponente (BYTE 0 ... 255).
2. G-Komponente (BYTE 0 ... 255).
3. B-Komponente (BYTE 0 ... 255).

5.1.5 peaking

Glyph

PALplus Toolbox Subform

Choose Selection

RAW TO CVBS

CVBS TO YC

C TO UV

COLORplus

HELPER

LINE 23

PEAKING

CTI

MEDIAN FILTER

YUV TO RGB

SAMPLING RATE

HELP

QUIT

Enhance the overall sharpness of Component Signals

Input Signal

Output Signal

Parameters for weighting of signals (0...100):

av ah

cv ch

Sampling Frequency (f_{sc} = Frequency of Color Carrier):

☒ $4 * f_{sc}$ ☐ $8 * f_{sc}$ ☐ $12 * f_{sc}$

Execute Quit Help

Abbildung 48: Programmoberfläche (GUI) von „peaking“

„peaking“ erhöht die Bildschärfe im Luminanzsignal [32].

- Eingabe:

1. VIFF Luminanz-Datei (BYTE 0 ... 255) bzw. (BYTE 16 ... 235).

- Parameter:

1. Parameter der nichtlinearen Kennlinien:
 - (a) a_v Verstärkung für den vertikalen Hochpaßanteil.
 - (b) c_v „Coring“ der Kennlinie für den vertikalen Hochpaßanteil.
 - (c) a_h Verstärkung für den horizontalen Hochpaßanteil.
 - (d) c_h „Coring“ der Kennlinie für den horizontalen Hochpaßanteil.
2. Abtastfrequenz der VIFF Luminanz-Datei (4, 8 oder 12-fache Farbträgerfrequenz).

- Ausgabe:

1. VIFF Luminanz-Datei (BYTE 0 ... 255) bzw. (BYTE 16 ... 235).

5.1.6 cti

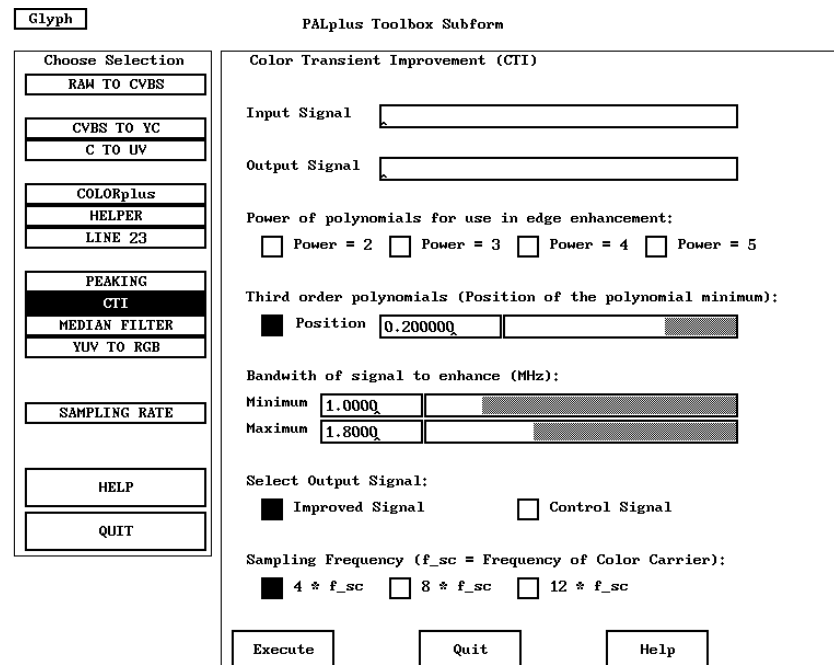


Abbildung 49: Programmoberfläche (GUI) von „cti“

„cti“ erhöht die Bildschärfe in den U/V-Komponenten [6]. Der „Color Transient Improvement“-Algorithmus detektiert Farbübergänge in den U/V-Komponenten, und ersetzt diese durch steilere Polynome.

- Eingabe:

1. VIFF U/V-Datei (BYTE 0 ... 255) bzw. (BYTE 128 ± 112).

- Parameter:

1. Grad des Polynoms (2 ... 5) zur Kantenversteilerung (kein Überschwingen).
2. Relative Position des Maxi-/Minimums beim Polynom dritten Grades.
3. Minimale und Maximale Bandbreite der Übergänge, die versteilert werden sollen.
4. Zur Überprüfung, welche Farbübergänge versteilert werden, kann das Kontrollsignal, anstatt des in der Schärfe verbesserten Farbdifferenzsignals, ausgegeben werden.
5. Abtastfrequenz der VIFF U/V-Datei (4, 8 oder 12-fache Farbträgerfrequenz).

- Ausgabe:

1. VIFF U/V-Datei (BYTE 0 ... 255) bzw. (BYTE 128 ± 112).

5.1.7 medfil

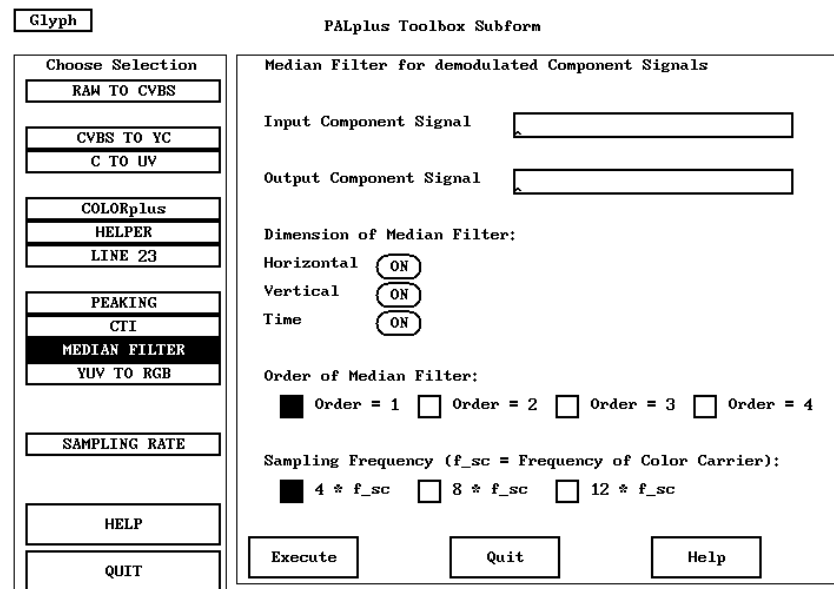


Abbildung 50: Programmoberfläche (GUI) von „medfil“

„medfil“ reduziert Rauschen, Cross-Color und Cross-Luminanz Störungen [38]. Das Medianfilter ist ein nichtlineares Filter mit tiefpaßähnlichem Verhalten, und besonderer Eignung zur Dämpfung impulsförmigen Rauschens. Aus einer Menge von benachbarten Bildpunkten, wird der mit der mittleren Amplitude selektiert. Die Filterung ist also ein Umsortieren, Wiederholen und Weglassen von Bildpunkten.

- Eingabe:

1. VIFF YUV-Datei (BYTE 0 ... 255).

- Parameter:

1. Dimension des Medianfilters: Die Dimension in der die Bildpunkte liegen, die in den Selektionsprozeß einbezogen werden sollen.
 - (a) Horizontal: In der selben Zeile, die benachbarten Bildpunkte.
 - (b) Vertical: Im selben Halbbild, die Bildpunkte in den darüber und darunter liegenden Zeilen.
 - (c) Time: In den Vollbildern davor und danach.
2. Ordnung des Median Filters (1 ... 4) gibt an, wieviele Bildpunkte in jeder der gewählten Dimensionen in den Selektionsprozeß einbezogen werden.
3. Abtastfrequenz der VIFF YUV-Datei (4, 8 oder 12-fache Farbträgerfrequenz).

- Ausgabe:

1. VIFF YUV-Datei (BYTE 0 ... 255).

5.1.8 colplus

Glyph PALplus Toolbox Subform

Choose Selection

RAW TO CVBS

CVBS TO YC

C TO UV

COLORplus

HELPER

LINE 23

PEAKING

CTI

MEDIAN FILTER

YUV TO RGB

SAMPLING RATE

HELP

QUIT

COLORplus-Decoder

Input CVBS Signal

Input Color Carrier

☐ Input Impulse Response (Y low)
☐ Input Impulse Response (Y high)
☐ Input Impulse Response (C low)
☐ Input Impulse Response (C high)

Output Y Component

Output U Component

Output V Component

☐ Output Motion Detection Signal

Select COLORplus Decoding:

☐ Simple COLORplus (without Motion Detection)
☒ Full COLORplus (with Motion Detection)

Parameters for Motion Detection:

☒ X-Dimension of Detection Matrix
☒ Y-Dimension of Detection Matrix
☒ Threshold for Hard Switching

Sampling Frequency (f_{sc} = Frequency of Color Carrier):

☒ $4 * f_{sc}$
 ☐ $8 * f_{sc}$
 ☐ $12 * f_{sc}$

PAL Delay-Line:

☒ Use PAL Delay-Line
 ☐ No PAL Delay-Line

Abbildung 51: Programmoberfläche (GUI) von „colplus“

„colplus“ ist ein Decoder, der nach dem COLORplus-Verfahren arbeitet [19], und als verbessertes Codiervorgang im Rahmen von PALplus eingesetzt wird. Die Vorteile von COLORplus gegenüber Standard-PAL sind:

1. Vermeidung von Cross-Color.
2. Vermeidung von Cross-Luminanz.
3. Ausnutzung der Kanalbandbreite von 5.6 MHz für die Luminanz.
4. Erhöhung der Farbbandbreite.

Zur Ausnutzung der verbesserten Eigenschaften des COLORplus-Decoders muß die zu decodierende VIFF CVBS-Datei COLORplus codiert sein. Dies bedeutet, daß sich die Chrominanzinformation und die hochfrequente Luminanzinformation im ersten und zweiten Halbbild eines Vollbildes jeweils wiederholen.

- Eingabe:

1. VIFF CVBS-Datei (BYTE 0 ... 255), wie sie „raw2cvbs“ liefert.
2. VIFF Farbträger-Datei (BYTE 128 ± 127), wie sie „raw2cvbs“ liefert.
3. Impulsantwort des Tiefpaßfilters (FLOAT) für die Luminanz (Y low). Die Anzahl der Koeffizienten des Filters muß ungeradzahlig sein (Grenzfrequenz: ≈ 3.0 MHz).
4. Impulsantwort des Bandpaßfilters (FLOAT) für die Luminanz (Y high). Die Anzahl der Koeffizienten des Filters muß ungeradzahlig sein (Durchlaßbereich: ≈ 3.0 MHz ... ≈ 6.0 MHz). Luminanztiefpaß und Luminanzbandpaß müssen spiegelsymmetrisch zur Übergangsfrequenz sein. Da Hoch- und Tiefpaßanteil nach der Bearbeitung addiert werden, muß die Dämpfung an der Übergangsfrequenz der Filter 6 dB betragen.
5. Impulsantwort des Tiefpaßfilters (FLOAT) für die demodulierten U/V-Komponenten (C low). Die Anzahl der Koeffizienten des Filters muß ungeradzahlig sein. Der Tiefpaß ist hinter die Demodulation geschaltet (Grenzfrequenz: ≈ 1.3 MHz).
6. Impulsantwort des Bandpaßfilters (FLOAT) zur Abtrennung der Chrominanz (C high). Die Anzahl der Koeffizienten des Filters muß ungeradzahlig sein. Der Bandpaß sollte symmetrisch zur Farbträgerfrequenz sein (Durchlaßbereich: $4.43 \text{ MHz} \pm \approx 1.3 \text{ MHz}$).

- Parameter:

1. COLORplus-Decodierung:
 - (a) Simple COLORplus: COLORplus-Decodierung ohne Bewegungserkennung.
 - (b) Full COLORplus: COLORplus-Decodierung mit Bewegungserkennung in den Farbdifferenzsignalen, und Umschaltung auf den „Fall-Back“-Modus. Der „Fall-Back“-Modus liefert eine hohe zeitliche Auflösung für die Farbe, und unterdrückt gleichzeitig die hochfrequente Luminanz.
2. Parameter für die Bewegungserkennung:
 - (a) X-Dimension: Breite der Matrix zur Bewegungserkennung.
 - (b) Y-Dimension: Höhe der Matrix zur Bewegungserkennung.
 - (c) Threshold: Schwellwert (0 ... 255) zur harten Umschaltung zwischen COLORplus und „Fall-Back“-Modus.
3. Abtastfrequenz der VIFF CVBS/Farbträger-Datei (4, 8 oder 12-fache Farbträgerfrequenz).
4. PAL Verzögerungsleitung ($64.057 \mu\text{s}$):
 - (a) Standard-PAL (Use PAL Delay-Line).
 - (b) Simple-PAL (No PAL Delay-Line).

- Ausgabe:

1. VIFF Y-Datei (BYTE 16 ... 235).
2. VIFF U-Datei (BYTE 128 $\pm$ 112).
3. VIFF V-Datei (BYTE 128 $\pm$ 112).
4. VIFF Motion Detection Signal (BYTE 0 ... 255). Die optionale Ausgabe dieser Datei dient der Parameterwahl für die Bewegungserkennung. Ihr Inhalt ist abhängig von den eingestellten Parametern. Wenn die X-oder/und Y-Dimension für die Matrix zur Bewegungserkennung abgeschaltet ist, so wird eine Matrix der Dimension 1 verwendet, was gerade einem Punkt entspricht. Ist der Schwellwert abgeschaltet, so werden die Daten zur Bewegungserkennung direkt weiterverwendet. Die Abschaltung des Schwellwerts ist nur dann sinnvoll, wenn die Datei zur Bewegungserkennung betrachtet werden soll, um für die eingestellte Matrix-Dimension einen günstigen Schwellwert festzulegen. Dies geschieht am besten unter Verwendung eines aussagekräftigen Testbildes, mit schnell bewegter, stark gesättigter Farbe. Für den normalen Betrieb im „Full COLORplus“-Modus sollte der Schwellwert immer eingeschaltet sein.

Nachfolgend sind die Filterkoeffizienten für die im Zusammenhang mit „colplus“ verwendeten FIR-Filter aufgelistet [29][23].

Filterkoeffizienten: Y low			Filterkoeffizienten: Y high		
H(1)	4.997666E-03	H(17)	H(1)	-1.937574E-02	H(17)
H(2)	1.714714E-02	H(16)	H(2)	5.107161E-05	H(16)
H(3)	1.035211E-02	H(15)	H(3)	-2.376997E-02	H(15)
H(4)	-2.391082E-02	H(14)	H(4)	3.436526E-05	H(14)
H(5)	-5.691721E-02	H(13)	H(5)	1.189797E-01	H(13)
H(6)	-2.058031E-02	H(12)	H(6)	5.021188E-05	H(12)
H(7)	1.146479E-01	H(11)	H(7)	-2.316711E-01	H(11)
H(8)	2.819929E-01	H(10)	H(8)	2.305560E-05	H(10)
H(9)	3.584880E-01		H(9)	2.827116E-01	

Filterkoeffizienten: C low			Filterkoeffizienten: C high		
H(1)	-2.725990E-03	H(19)	H(1)	-4.385440E-02	H(17)
H(2)	-6.937651E-03	H(18)	H(2)	2.624489E-05	H(16)
H(3)	-1.069060E-02	H(17)	H(3)	6.055354E-04	H(15)
H(4)	-8.374887E-03	H(16)	H(4)	2.125975E-05	H(14)
H(5)	6.658392E-03	H(15)	H(5)	1.156994E-01	H(13)
H(6)	3.834685E-02	H(14)	H(6)	-9.381519E-06	H(12)
H(7)	8.409808E-02	H(13)	H(7)	-2.631362E-01	H(11)
H(8)	1.336817E-01	H(12)	H(8)	-2.463572E-05	H(10)
H(9)	1.723185E-01	H(11)	H(9)	3.313270E-01	
H(10)	1.869338E-01				

5.1.9 line23

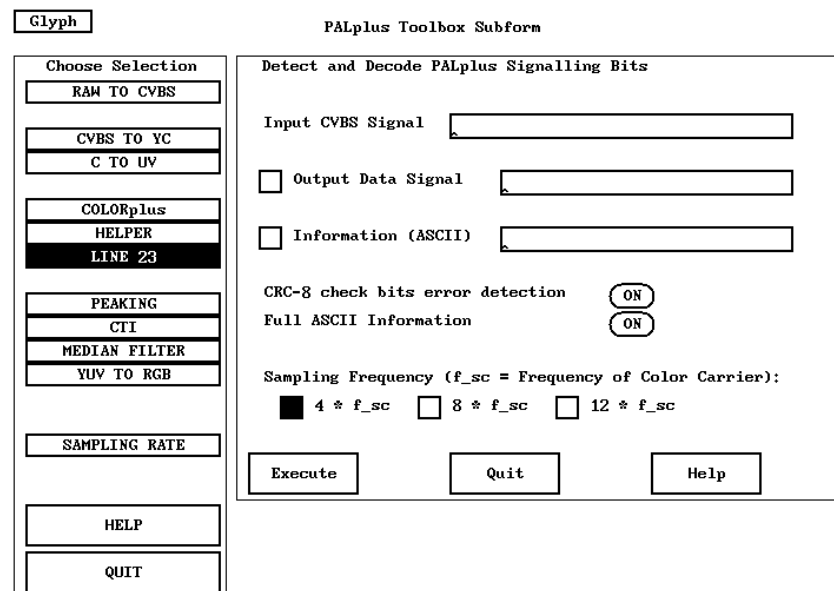


Abbildung 52: Programmoberfläche (GUI) von „line23“

„line23“ decodiert die Daten-Bits der Zeile 23 in jedem Vollbild. Wenn sich die Information von einem Vollbild zum nächsten ändert, wird eine Warnung ausgegeben.

- Eingabe:
 1. VIFF CVBS-Datei (BYTE 0 ... 255), wie sie „raw2cvbs“ liefert.
- Parameter:
 1. CRC-8: Wenn der Schalter auf „ON“ steht, so wird eine Fehlererkennung mit dem CRC-8 Polynom durchgeführt.
 2. Full ASCII: Wahlweise läßt sich hier auf eine erweiterte Ausgabe der ASCII-Information umschalten. Es werden dann auch die Bits in der Biphas-L Codierung ausgegeben.
 3. Abtastfrequenz der VIFF CVBS-Datei (4, 8 oder 12-fache Farbträgerfrequenz).
- Ausgabe:
 1. VIFF Data-Datei (BYTE 0/1). Enthält 33 Bytes, welche die 32 Data-Bits (b00 ... b31) und das WSL-Bit repräsentieren.
 2. Information (ASCII). Diese Ausgabedatei kann mit dem „ASCII File Viewer“ unter KHOROS betrachtet werden. Sie enthält alle Informationen über die Daten-Bits im Klartext. Wenn der Ausgang nicht belegt ist, so wird die Information nach „stdout“ geschrieben.

5.1.10 helper

Abbildung 53: Programmoberfläche (GUI) von „helper“

„helper“ demoduliert den vertikalen Helper in einer PALplus CVBS-Datei.

- Eingabe:

1. VIFF CVBS-Datei (BYTE 0 ... 255), wie sie „raw2cvbs“ liefert.
2. Impulsantwort eines Tiefpaßfilters (FLOAT). Die Anzahl der Koeffizienten des Filters muß ungeradzahlig sein. Der Tiefpaß ist direkt hinter den Demodulator geschaltet (Grenzfrequenz: ≈ 4.0 MHz).

- Parameter:

1. Phasen-Offset ($0^\circ \dots 360^\circ$) zur relativen Farbträgerphase 0° .
2. „Coring“ (0 ... 1) vor der nichtlinearen Kennlinie. Alle Werte die kleiner sind als der eingestellte Wert, werden gleich 0 gesetzt.
3. „Clipping“ (0 ... 1) nach der nichtlinearen Kennlinie. Alle Werte die größer als der eingestellte Wert sind, werden gleich diesem Wert gesetzt.
4. Das Polynom zur nichtlinearen Gewichtung ist von der Form $a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$. Da das genaue Aussehen der nichtlinearen Kennlinie im PALplus-Decoder nicht bekannt ist, kann hier jede mögliche Kennlinie mit einem Polynom fünften Grades angenähert werden.

5. Verstärkung des Helpersignals nach „Clipping“.
 6. Abtastfrequenz der VIFF CVBS-Datei (4, 8 oder 12-fache Farbträgerfrequenz).
- Ausgabe:
 1. VIFF Helper-Datei (BYTE 128 ± 127).

5.2 PAL-Workspace

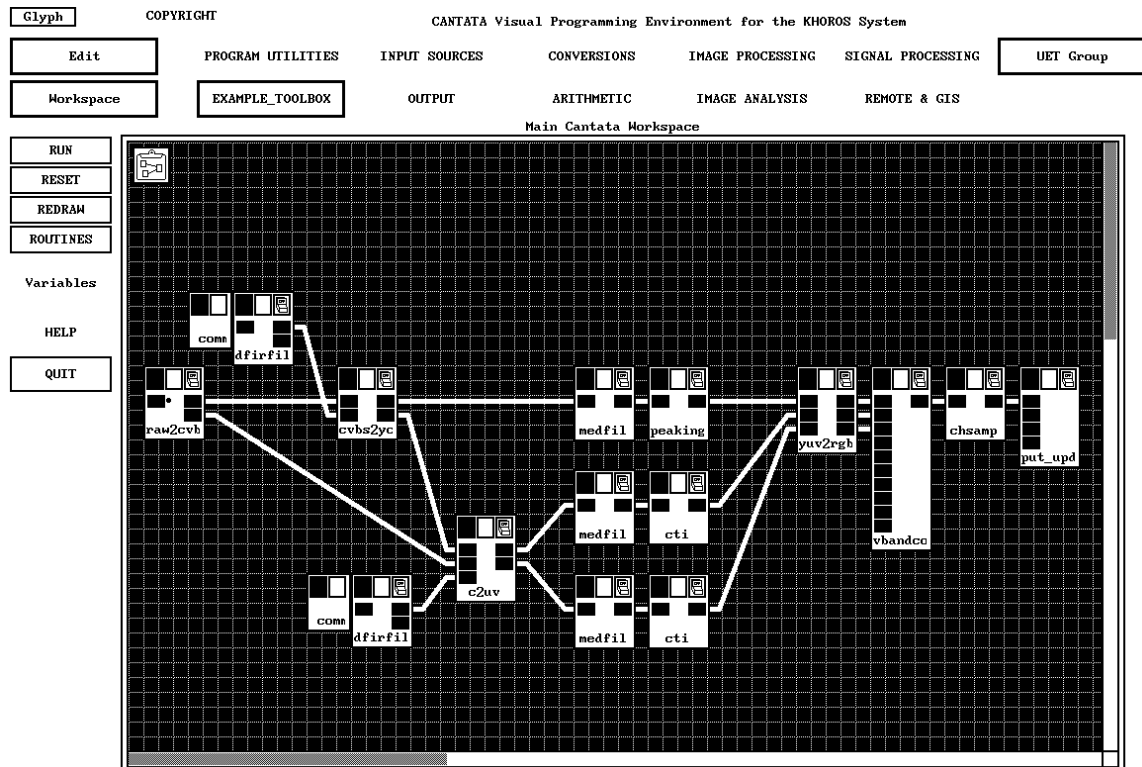


Abbildung 54: Workspace des PAL-Decoders

Die in Kapitel 5 beschriebenen Programme simulieren die einzelnen Funktionsgruppen eines PAL/PALplus-Decoders. Unter KHOROS können diese Programme zu Blockschaltungen verbunden werden. Die graphische Simulationsoberfläche „cantata“ stellt alle Funktionen zum Aufbau solcher Schaltungen bereit. Die Bedienung von „cantata“ erfolgt dabei völlig intuitiv. Durch das simple Anklicken eines Ausgangs und eines Eingangs wird etwa eine Verbindung zwischen diesen erzeugt, und mit einer Verbindungslinie graphisch angezeigt. Ein Klick auf eine Verbindungslinie ermöglicht ihre Trennung. Die durch „glyphen“ symbolisierten Programme können mit der Maus bedient und verschoben werden. Eine solche, mit „cantata“ erstellte Blockschaltung nennt sich „workspace“. Anschließend folgt die Beschreibung des „workspace“ für den PAL-Decoder.

Die Simulation des PAL-Decoders beginnt mit „raw2cvbs“. Dieses Programm liest ein „rohes“ CVBS-Signal ein und sucht, nach der PAL B/G Norm, den Anfang eines Halbbildes mit ungerader Nummer. Danach wird die Phase des Burstes und seine Lage bestimmt, die Aufschluß darüber gibt, ob das erste Halbbild die Nummer 1 oder 3 hat. Die skalierte VIFF CVBS-Datei wird an „cvbs2yc“ weitergegeben, während der regenerierte Farbträger zu „c2uv“ geleitet wird. „cvbs2yc“ trennt die Luminanz und die Chrominanz, mit verschiedenen einstellbaren Verfahren, voneinander. Hierzu muß allerdings die Impulsantwort eines Bandpaßfilters, wie es z.B. mit „dfirfil“ [23] entworfen werden kann, an „cvbs2yc“ angeschlossen sein. Die Lu-

minanz aus „cvbs2yc“ ist skaliert nach CCIR-Rec.601 [24], und kann direkt auf die Decoder-Matrix „yuv2rgb“ geleitet werden. In dem in Abbildung 5.2 dargestellten Beispiel durchläuft die Luminanz jedoch noch ein Medianfilter, um Cross-Luminanz und Rauschstörungen zu dämpfen. Weiterhin wird die Bildschärfe der Luminanz mit dem Programm „peaking“ verbessert. Die Chrominanz aus „cvbs2yc“ gelangt zur Demodulation zum Programm „c2uv“. Hierbei handelt es sich um einen Synchron-demodulator mit nachgeschaltetem Tiefpaß. Wahlweise kann „c2uv“ auf Simple-PAL oder auf Standard-PAL umgeschaltet werden. Die U/V-Komponenten werden nach CCIR-Rec.601 skaliert, und können ebenfalls direkt an „yuv2rgb“ angeschlossen werden. Im gezeigten Beispiel durchlaufen aber auch sie je ein Medianfilter, zur Dämpfung von Cross-Color Störungen. Die nachfolgenden „cti“-Programme erhöhen die Bildschärfe der bandbegrenzten U/V-Komponenten mit einem nichtlinearen Verfahren. In der Decoder-Matrix „yuv2rgb“ werden die nach CCIR-Rec.601 skalierten YUV-Komponenten in RGB-Signale umgerechnet. Anschließend werden die 3 Dateien in „vbandcomb“ zu einer Datei zusammengefaßt. Das Programm „chsamp“ ändert die Abtastfrequenz von $17.73 \text{ MHz} (4 \cdot f_{sc})$ auf $13.5 \text{ MHz} (864 \cdot f_h)$. Die resultierende Datei kann dann unter KHOROS mit „put_update“ betrachtet werden.

5.3 PALplus-Workspace

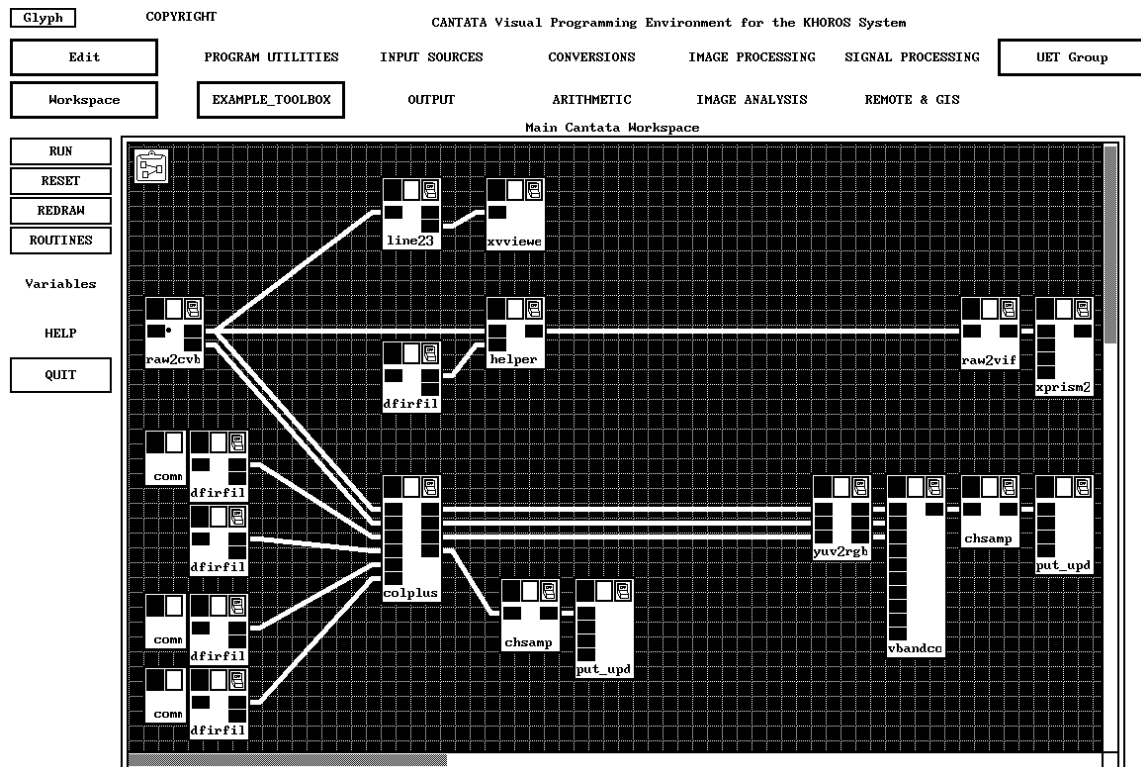


Abbildung 55: Workspace des PALplus-Decoders

Den Anfang in der Simulationsumgebung für den PALplus-Decoder (Abbildung 5.3) macht, wie im PAL-Decoder (Abbildung 5.2), das Programm „raw2cvbs“. Es

liefert eine skalierte VIFF CVBS-Datei und eine Farbträgerdatei, die an „colplus“ weitergegeben werden. „colplus“ beinhaltet einen COLORplus-Decoder, mit frei einstellbaren Parametern. Zu seiner Funktion müssen noch die Impulsantworten von vier verschiedenen Filtern angeschlossen werden (siehe Kapitel 5.1.8). Die YUV-Komponenten am Ausgang von „colplus“ sind nach CCIR-Rec.601 skaliert, und können direkt auf die Decoder-Matrix „yuv2rgb“ gegeben werden. Die VIFF CVBS-Datei wird außerdem zum Status-Decoder „line23“ geleitet. Hier erfolgt die Demodulation der Signalisierungsbits in Zeile 23 eines jeden Vollbildes. Die Signalisierungsdatei des Ausgangs wird von einem Programm, das den Quadratur-Spiegelfilter (QMF) zur Bandsynthese enthält, benötigt. Die Ausgabe der in den Daten-Bits codierten Information, kann mit dem ASCII File Viewer „xvviewer“ erfolgen. Das Programm „helper“ erwartet ebenfalls die VIFF CVBS-Datei am Eingang, und führt die Demodulation des vertikalen Helpers durch. Alle Parameter zur Entzerrung und Skalierung des Helpers können dabei frei definiert werden.

Die Ausgangsdatei aus „helper“, mit der hochfrequenten Luminanz, und die tief-frequente Luminanz aus „colplus“, müssen zur Decodierung nach dem PALplus-Verfahren noch die Bandsynthese in einem QMF durchlaufen. Die U/V-Komponenten aus „colplus“ werden abschließend von 432 auf 576 Zeilen vertikal interpoliert. Sowohl das Quadratur-Spiegelfilter, als auch die vertikale Interpolation sind in diesem „workspace“ noch nicht enthalten [21].

Die resultierenden YUV-Komponenten werden dann in der Dematrix „yuv2rgb“ in RGB-Signale matriziert und können, nach der Zusammenfassung mit „vband-comb“, mit „put_update“ betrachtet werden.

6 Zusammenfassung

Ziel der vorliegenden Arbeit war es, Programme zur Simulation eines PAL-Decoders und eines PALplus-Decoders unter KHOROS (UNIX) zu erstellen. Zu diesem Zweck wurde der PAL/PALplus-Decoder in einzelne Funktionsgruppen unterteilt, die getrennt realisiert werden konnten.

Der erste Schritt zur Entwicklung des Simulationspakets war die Implementierung eines digitalen Standard PAL-Decoders. Beginnend mit der Detektion des Halbbildanfangs und der Rückgewinnung des Farbträgers zur Farbdemodulation wurde das Programm „raw2cvbs“ entwickelt, das die digitalen CVBS-Signale einer D3 Bandmaschine in eine für KHOROS lesbare VIFF CVBS-Datei konvertiert. Zur Demodulation der U/V-Komponenten wurde das Programm „c2uv“ erstellt, das die Umschaltung zwischen Simple-PAL und Standard-PAL erlaubt. Die Trennung von Luminanz und Chrominanz erfolgte über einfache Notch- und Bandpaßfilter. Eine wesentliche Verbesserung wurde durch die Einführung des adaptiven Bild- und Zeilenkammfilters „cvbs2yc“ erreicht. Mit dem Kammfilter wurde die nutzbare Luminanz- und Chrominanzbandbreite erheblich vergrößert. Weiterhin konnten damit die Cross-Luminanz und Cross-Color Störungen entscheidend verringert werden. Zur Elimination von Rauschstörungen und verbleibenden Cross-Störungen kann ein Medianfilter für die YUV-Komponenten zugeschaltet werden (Programm: „medfil“). Das Medianfilter bewirkt eine sichtbare Erhöhung des Signal/Geräuschabstands, ohne die Bildschärfe zu beeinträchtigen. Zum Ausgleich des aperturbedingten Frequenzgangabfalls im Luminanzkanal, wurde eine horizontal und vertikal arbeitende Peakingschaltung implementiert (Programm: „peaking“). Hierdurch kann die Bildschärfe der Luminanz stark verbessert werden. Die begrenzte Bandbreite der Farbdifferenzsignale verbietet jedoch die Anwendung einer derartigen Peakingschaltung für die U/V-Komponenten. Um auch hier eine Erhöhung der Schärfe zu erreichen, wurde ein nichtlinearer Algorithmus zur Kantenversteigerung an Farbübergängen programmiert (Programm: „cti“). Das CTI-Verfahren (Color Transient Improvement) detektiert Farbübergänge, deren Länge in den Bereich der Filterbandbreite fällt, und ersetzt sie durch steilere Polynomfunktionen. Zur Umrechnung der YUV-Komponenten in RGB-Signale kann die Dematrix des Programms „yuv2rgb“ genutzt werden.

Das COLORplus-Verfahren stellt bei der Codierung und Decodierung eine kompatible Verbesserung des PAL-Standards dar. Als Teil von PALplus wird damit eine Erhöhung der Luminanz- und Farbbandbreite, bei gleichzeitiger Vermeidung von Cross-Color und Cross-Luminanz Störungen erreicht. Das Programm „colplus“ beinhaltet einen solchen Decoder. Zuerst wurde ein Simple COLORplus Verfahren programmiert, das jedoch die bekannten Nachteile von COLORplus beinhaltet. So setzt, durch die verminderte zeitliche Auflösung der Chrominanz, bei schnell bewegten und stark gesättigten Farben, ein störender Ruckeleffekt ein. Die Weiterentwicklung zu Full COLORplus beinhaltet demzufolge eine Bewegungserkennung für solche kritischen Bildbereiche, und schaltet dort in den „Fall-Back“-Modus um, der eine hohe zeitliche Auflösung für die Farbe bietet.

Ein weiteres Merkmal von PALplus ist die Option auf ein 16:9 Bildformat. PALplus bedient sich dabei der Letterbox-Codierung zur kompatiblen Übertragung des

16:9 Bildes im terrestrische Kanal. Die hierfür speziell codierte Zusatzinformation, der vertikale Helper, am oberen und unteren Bildbereich kann mit dem Programm „helper“ demoduliert und entzerrt werden.

Weiterhin beinhaltet die PALplus Spezifikation die Übertragung von Daten-Bits in der Zeile 23. Sie dienen der Kenntlichmachung des Codermodes, über das Bildformat und die Quellenart (Film oder Kamera), im Decoder. Zur Decodierung der Daten-Bits enthält das entwickelte Simulationspaket einen Status-Decoder, das Programm „line23“.

Die Programme zur Bandsynthese der vertikal aufgespaltenen Luminanz im 16:9 Empfänger, und zur vertikalen Interpolation der U/V-Komponenten sind noch nicht verfügbar [21].

Mit dem entwickelten PAL/PALplus-Simulationspaket besteht die Möglichkeit, diverse Algorithmen zur Bildverarbeitung im Decoder, mit unterschiedlichen Parametern zu testen und zu bewerten. Mit den Programmen „medfil“, „peaking“ und „cti“ wurden Algorithmen implementiert, die eine wesentliche Verbesserung der Bildqualität, im Vergleich zu handelsüblichen Empfängern, garantieren. Zur Simulation von PALplus sind die Programme mit nicht bekannten Parametern auf freie Konfiguration durch den Benutzer ausgelegt.

Literatur

- [1] ACHIHA, M.: *Bewegungsadaptiver Qualitätsdecoder und Up-Konverter für PAL-Signale*, (deutsche Bearbeitung von H. Gondolf). Fernseh- und Kino-Technik, 46(2):98–104, 1992.
- [2] ARGIRO, D., C. GAGE, J. RASURE, M. YOUNG, T. SAUER, S. JORGENSEN und C. WILLIAMS: *KHOROS Volume I (User's Manual) and Volume II (Programmer's Manual)*. The KHOROS Group, Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque NM 87131, 1 Auflage, 1992.
- [3] CAMERA, FAIRCHILD und INSTRUMENTS: *Collection of Applications*. te-wi Verlag GmbH, München, Deutschland, 1 Auflage, 1978.
- [4] CROCHIERE, R.E. und L.R. RABINER: *Multirate Digital Signal Processing*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1 Auflage, 1983.
- [5] EBNER, A., E. MATZEL, R. MORCOM, U. RIEMANN, M. SILVERBERG, R. STOREY, F. VREESWIJK und D. WESTERKAMP: *PALplus: Übertragung von 16:9-Bildern im terrestrischen PAL-Kanal*. Fernseh- und Kino-Technik, 46(11):733–739, 1992.
- [6] FONSALAS, F. und J.Y. LEJARD: *An Edge Enhancement Algorithm using a Polynomial Transformation*. IEEE Transactions on Consumer Electronics, 32(3):151–154, 1986.
- [7] GRAFE, T. und G. 17 SCHEFFLER: *Interfield Noise and Cross Color Reduction IC for Flicker Free TV Receivers*. IEEE Transactions on Consumer Electronics, 34(3):402–409, August 1988.
- [8] HEBER, H., E. WINTER und A. ZIEMER: *Neue Studios für verbesserte Fernsehnormen*. Fernseh- und Kino-Technik, 45(1):13–19, 1991.
- [9] HENTSCHEL, C.: *Bandaufspaltung zur Rasterkonversion für eine PALplus-Übertragung*. Fernseh- und Kino-Technik, 46(11):742–754, 1992.
- [10] HENTSCHEL, C., C. JOHANSEN und D. TEICHNER: *Bildspeichergestützte digitale Verarbeitung von Farbfernsehsignalen*. Fernseh- und Kino-Technik, 40(3):112–117, 1986.
- [11] HENTSCHEL, C., C. JOHANSEN und D. TEICHNER: *Bildspeichergestützte digitale Verarbeitung von Farbfernsehsignalen (Fortsetzung)*. Fernseh- und Kino-Technik, 40(4):152–156, 1986.
- [12] HENTSCHEL, C. und H. SCHÖNFELDER: *Probleme des Formatwechsels bei zukünftigen verbesserten PAL-Verfahren*. Fernseh- und Kino-Technik, 45(7):347–355, 1991.

- [13] HERFET, T.: *Bandaufspaltung zur kompatiblen 16:9-Übertragung*. Rundfunktechnische Mitteilungen, 35(1):29–35, 1991.
- [14] HERFET, T.: *Breitbild-PAL mit vertikaler Bandaufspaltung*. Fernseh- und Kino-Technik, 46(10):673–679, 1992.
- [15] INTERNATIONAL RADIO CONSULTATIVE COMMITTEE (CCIR): *Recommendations and Reports of the CCIR (XVIth Plenary Assembly)*, Dubrovnik, 1986. International Telecommunication Union (ITU). Volume XI - Part 1.
- [16] ITO, SHIN-ETSU, R. LOWLOR und N. EBIHARA: *Development of a crosstalk measurement type three-dimensional digital color decoder*. IEEE Transactions on Consumer Electronics, 39(2):63–70, Mai 1993.
- [17] JEON, J.H. und J.K. KIM: *New linear phase QMF filter design for sub-band coding*. IEEE Electronics Letters, 27(4):319–320, Februar 1991.
- [18] KAYS, R.: *Bildqualitätsverbesserung im PAL-Standard*. Fernseh- und Kino-Technik, 44(4):181–184, 1990.
- [19] KAYS, R.: *Ein Verfahren zur verbesserten PAL-Codierung und -Decodierung*. Fernseh- und Kino-Technik, 44(11):595–602, 1990.
- [20] KÖHNE, H.: *2-H-Adaptive Combfilter Video Processor*. IEEE Transactions on Consumer Electronics, 37(3):303–308, August 1991.
- [21] KÜTHE, H.: *Entwicklung von Programmen zur Simulation der Formatkonversion in PALplus-Empfängern*, 1993. (D 2815 ÜT) Technische Hochschule Darmstadt.
- [22] MALVAR, H.S.: *Modulated QMF filter banks with perfect reconstruction*. IEEE Electronics Letters, 26(13):906–907, Juni 1990.
- [23] MCCLELLAN, J.H., T.W. PARKS und L.R. RABINER: *A Computer Program for Designing Optimum FIR Linear Phase Digital Filters*. IEEE Transactions on Audio and Electroacoustics, 21(6):506–526, Dezember 1973.
- [24] NÄTHER, N. und H. SCHWARZ: *Der digitale TV Markt – Evolution des 4:2:2-Standards*. Fernseh- und Kino-Technik, 43(3):131–134, 1989.
- [25] PERLMAN, STUART S. und R. SCHWEER: *An Adaptive Luma-Chroma Separator circuit for PAL and NTSC TV Signals*. IEEE Transactions on Consumer Electronics, 36(3):301–307, 1990.
- [26] PETERSON, W.W.: *Prüfbare und korrigierbare Codes*. R. Oldenbourg Verlag, München-Wien, 1967.
- [27] REIMERS, U., C.P. SANDBANK und A. ZIEMER: *PALplus – eine vollkompatible Weiterentwicklung des PAL-Farbfernsehens*. Fernseh- und Kino-Technik, 45(8):391–397, 1991.

- [28] RICHTER, H.P.: *Komponentensignale in der Videotechnik – Ein Grundlagen-Beitrag*. Fernseh- und Kino-Technik, 45(6):285–292, 1991.
- [29] RUPPEL, W.: *FIR-Filter für COLORplus*, Oktober 1993.
- [30] SCHÖNFELDER, H.: *Fernsehtechnik Teil 1 und Teil 2*. Justus von Liebig Verlag, Darmstadt, 1973.
- [31] SCHÖNFELDER, H.: *Bildkommunikation*. Springer Verlag, Berlin, Heidelberg, New York, 1983.
- [32] SCHÖNFELDER, H.: *Digitale Filter der Videotechnik*. Drei-R-Verlag, Berlin, 1. Auflage, 1988.
- [33] SCHWEER, R. und M. PLANTHOLT: *Adaptive Luminanz-Chrominanz-Trennung mit unterschiedlich farbträgerverkoppelten Samplingphasen*. Fernseh- und Kino-Technik, 44(12):673–681, 1990.
- [34] TEICHNER, D.: *Adaptive Filter Techniques for Separation of Luminance and Chrominance in PAL TV Signals*. IEEE Transactions on Consumer Electronics, 32(3):241–250, August 1986.
- [35] TEICHNER, D.: *PAL-Coder und -Decoder mit dreidimensionalen Filtertechniken - Subjektive Tests*. Fernseh- und Kino-Technik, 43(6):310–322, 1986.
- [36] TEICHNER, D.: *Qualitätsverbesserungen durch adaptive Inter-/Intraframe-Verarbeitung im PAL-Heimempfänger*. Fernseh- und Kino-Technik, 40(2):65–73, 1986.
- [37] TEICHNER, D.: *PAL-Coder und -Decoder mit dreidimensionalen Filtertechniken*. Fernseh- und Kino-Technik, 42(9):403–422, 1988.
- [38] WISCHERMANN, G.: *Medianfilterung von Videosignalen – Eine wirkungsvolle Alternative*. Fernseh- und Kino-Technik, 44(12):665–672, 1990.
- [39] ZIEMER, A. und E. MATZEL: *Der Weg zu PALplus - Eine kompatible Verbesserung des PAL-Systems*. Fernseh- und Kino-Technik, 43(8):407–410, 1989.

A Programmlistings

A.1 raw2cvbs

A.1.1 raw2cvbs.pane

```
-F 4.2 1 0 170x 7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1      100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +0+0 'Creates a CVBS VIFF signal from a raw CVBS file' raw2cvbs
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input Raw CVBS File' 'Input Raw CVBS File' i
-O 1 0 0 1 0 0 50x1+2+4 +0+0 ' ' 'Output CVBS Signal' 'Output CVBS Signal' o1
-O 1 0 0 1 0 0 50x1+2+6 +0+0 ' ' 'Output Color Carrier' 'Output Color Carrier' o2
-i 1 0 0 1 0 0 50x1+2+8 +0+0 0 256 0 'Skip Frames' 'Skip Frames' skip
-i 1 0 0 1 0 50x1+2+10 +0+0 1 256 4 'Frames in Output Signal' 'Number of Frames' frames
-T 1 0 0 1 0 40x1+2+12 +0+0 2 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 1 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-H 1 10x2+39+15 'Help' 'man page for raw2cvbs' $PALPLUS/doc/manpages/raw2cvbs.1
-Q 1 0 10x2+20+15 'Quit'
-R 1 0 1 10x2+1+15 'Execute' 'do operation' $PALPLUS/bin/raw2cvbs
-E
-E
-E
```

A.1.2 raw2cvbs.prog

```
-authors
Harald Zottmann
-authors_end

-short_prog_description
Creates a CVBS VIFF signal from a raw CVBS file.
-short_prog_description_end

-short_lib_description
Creates a CVBS VIFF signal from a raw CVBS file.
-short_lib_description_end

-mani_long_description
.I raw2cvbs
will create a CVBS VIFF signal from a raw CVBS file, and
a color subcarrier VIFF signal by refreshing the carrier
from the burst.
.LP
The program traces for the sync-signals between two
fields. Starting with the field-synchronizing datum of
an odd-numbered field, the CVBS-Signal will be scaled
and sent to the first output. Output two provides
the refreshed color subcarrier with a relative phase shift
of 0 degrees for following demodulation.
-mani_long_description_end

-mani_examples
raw2cvbs -i raw_cvbs.dat -o1 cvbs_outimage
-o2 carrier_outimage -fields 2 -f_sc 1
.LP
will generate cvbs_outimage and carrier_outimage from
raw_cvbs.dat. The outputimages will contain two fields.
The samplefrequency is set to the carrier-frequency
multiplied with four.
-mani_examples_end

-mani_restrictions
```

```
.I raw2cvbs
.LP
Datatypes of input and output signals:
.LP
input raw cvbs file:    1_BYTE (0...255)
.LP
output cvbs signal:     1_BYTE (0...255)
output color carrier:    1_BYTE (0...255)
                        (128-127...128+127)

The minimum level allowed for BLACK in the input
raw cvbs file is 16. The sync-level has to be 0.
.LP
The output cvbs signal is scaled:
.LP
WHITE + CHROMINANCE:  255
WHITE:                 207
BLACK:                 62
SYNC:                  0
-man1_restrictions_end

-man1_see_also
-man1_see_also_end

-man3_long_description
.I raw2cvbs
will create a CVBS VIFF signal from a raw CVBS file, and
a color subcarrier VIFF signal by refreshing the carrier
from the burst.
.LP
The program traces for the sync-signals between two
fields. Starting with the field-synchronizing datum of
an odd-numbered field, the CVBS-Signal will be scaled
and sent to the first output. Output two provides
the refreshed color subcarrier with a relative phase shift
of 0 degrees for following demodulation.
-man3_long_description_end

-man3_restrictions
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

-usage_additions
-usage_additions_end

-include_includes
-include_includes_end

-include_additions
-include_additions_end

-include_macros
-include_macros_end

-main_variable_list
kfile          *pal_file = NULL;

struct xvimage  *cvbs_img,
                *carrier_img;
-main_variable_list_end

-main_before_lib_call
if (check_args()) exit(1);

pal_file = klopen (raw2cvbs->i_file, "r");
if (pal_file == NULL)
{
(void) fprintf(stderr,"raw2cvbs : cannot read input file");\

```

```

        (void) fprintf(stderr, " %s\n", raw2cvbs->i_file);\
        exit(1);\
    }\
-main_before_lib_call_end

-main_library_call
    if(! lraw2cvbs(pal_file,
                   &cvbs_img, &carrier_img,
                   raw2cvbs->skip_int,
                   raw2cvbs->frames_int,
                   raw2cvbs->f_sc_toggle) )
    {
        fprintf(stderr, "lraw2cvbs failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if(! writeimage(raw2cvbs->o1_file, cvbs_img))
    {
        (void) fprintf(stderr, "raw2cvbs : unable to write cvbs output file\n");
        exit(1);
    }
    if(! writeimage(raw2cvbs->o2_file, carrier_img))
    {
        (void) fprintf(stderr, "raw2cvbs : unable to write carrier output file\n");
        exit(1);
    }
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lraw2cvbs( pal_file,
               cvbs_img,
               carrier_img,
               skip_frames,
               num_frames,
               f_sc_toggle)

kfile          *pal_file;

struct ximage   **cvbs_img,
                **carrier_img;

int             skip_frames,
                num_frames,
                f_sc_toggle;
-library_def_end

-library_code
/*****
/*
/* Library Code for raw2cvbs
/*
/*
*****/

#include "vinclude.h"
#include "math.h"

/*****

#define FCARRIER          4433618.75

```

```

#define FROW                15625

#define SYNCTHRESH          16
#define MAXBUF              16384

#define FC_FAC_4             1
#define FC_FAC_8             2
#define FC_FAC_12            3

#define NO_BURST             100.0

/*****
/*
/* Main Function
/*
/*
*****/
{
    long          searchbegin    ();
    double        getburstphase  ();

    signed long    len_pal_file   = 0L,
                  cvbs_start     = 0L,
                  skip_bytes     = 0L,
                  total_bytes    = 0L,
                  field_bytes    = 0L,
                  frame_bytes    = 0L,
                  il;

    signed int     i,j,k;
    unsigned char  *pal_ptr       = NULL,      /* Pointer to Raw-CVBS-Signal */
                  *sig_ptr       = NULL,      /* Pointer to CVBS-Signal */
                  *cvbs_ptr      = NULL,      /* Pointer to CVBS Imagedata */
                  *carrier_ptr   = NULL,      /* Pointer to Carrier Imagedata */
                  black_level    = 0;         /* Level of black */

    double         signal         = 0.0,
                  carrier_fac     = 0.0,      /* Multiplier */
                  samp_freq       = 0.0,      /* Sampling frequency */
                  phase           = 0.0,      /* Phase of color subcarrier */
                  omega           = 0.0,
                  adjust          = 0.0;      /* Scale CVBS signal */

/*****

    fprintf(stdout, "\nRaw2CVBS V1.3:\n");

    switch (f_sc_toggle)
    {
        case FC_FAC_4:
            carrier_fac = 4.0;
            break;

        case FC_FAC_8:
            carrier_fac = 8.0;
            break;

        case FC_FAC_12:
            carrier_fac = 12.0;
            break;

        default:
            fprintf (stderr, "Missing or improper argument in call to raw2cvbs.\n");
            return (0);
    }
    samp_freq = FCARRIER * carrier_fac;

/*****
/* Duration of one field 1/50 sec = 0.02 sec
/* Duration of one frame 1/25 sec = 0.04 sec
*****/

    field_bytes = (long) (0.02 * samp_freq);

```

```

frame_bytes = (long) (0.04 * samp_freq);
total_bytes = (long) (0.04 * samp_freq * (float) num_frames);
skip_bytes = (long) (0.04 * samp_freq * (float) skip_frames);

/*****
/* Read Raw-CVBS-Signal
*****/

kfseek (pal_file, 0L, SEEK_END);
len_pal_file = kftell (pal_file);
if (len_pal_file < frame_bytes)
{
    fprintf (stderr, "Not a single frame in raw CVBS signal.\n");
    return (0);
}
pal_ptr = kcalloc (len_pal_file, sizeof (unsigned char));
if (pal_ptr == NULL)
{
    fprintf (stderr, "Memory allocation failed.\n");
    return (0);
}

i = (signed int) (len_pal_file / MAXBUF);
j = (signed int) (len_pal_file % MAXBUF);

krewind (pal_file);
for (k = 0; k < i; k++)
{
    kfread ((pal_ptr + (k * MAXBUF)), MAXBUF, sizeof (unsigned char),
            pal_file);
}
if (j > 0)
{
    kfread ((pal_ptr + (i * MAXBUF)), j, sizeof (unsigned char),
            pal_file);
}

/*****
/* Search start of the first field and set pointer to it (sig_ptr)
*****/

cvbs_start = searchbegin (pal_ptr, len_pal_file, carrier_fac, &black_level);
if (cvbs_start == -1L)
{
    fprintf (stderr, "No sync found in raw CVBS signal.\n");
    return (0);
}
sig_ptr = (unsigned char *) (pal_ptr + cvbs_start);

/*****

if (black_level > SYNCTHRESH)
{
    adjust = (double) (43.0 / (double) black_level);
    adjust *= 255.0 / 176.0;
}
else
{
    fprintf (stderr, "Blacklevel in raw CVBS signal is too low.\n");
    return (0);
}

/*****

if ((len_pal_file - cvbs_start - skip_bytes) < total_bytes)
{
    fprintf (stderr, "Not enough frames in raw CVBS signal.\n");
    fprintf (stderr, "Only %d frames available.\n",
            (int) ((len_pal_file - cvbs_start) / (frame_bytes + 1L)));
    if (num_frames > 1)

```

```

        {
            fprintf (stderr, "Selected frames: %d - %d\n", skip_frames + 1,
                    skip_frames + num_frames);
        }
    else
    {
        fprintf (stderr, "Selected frame: %d\n", skip_frames + 1);
    }
    return (0);
}

if (num_frames > 1)
{
    fprintf (stdout, "Creating frames: %d - %d\n", skip_frames + 1,
            skip_frames + num_frames);
}
else
{
    fprintf (stdout, "Creating frame: %d\n", skip_frames + 1);
}

/*****
/* Allocate memory for Output-Images */
*****/

*cvbs_img = createimage ((unsigned long) (total_bytes), 1L, VFF_TYP_1_BYTE,
                        1,1, "cvbs_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
                        VFF_LOC_IMPLICIT, 0);

if (*cvbs_img == NULL)
{
    fprintf (stderr, "Out of memory.\n");
    return (0);
}
cvbs_ptr = (unsigned char *) (*cvbs_img)->imagedata;

*carrier_img = createimage ((unsigned long) total_bytes, 1L, VFF_TYP_1_BYTE,
                        1, 1, "color_carrier_signal", 0, 0, VFF_MS_NONE,
                        VFF_MAPTYP_NONE, VFF_LOC_IMPLICIT, 0);

if (*carrier_img == NULL)
{
    fprintf (stderr, "Out of memory.\n");
    return (0);
}
carrier_ptr = (unsigned char *) (*carrier_img)->imagedata;

/*****
/* Create CVBS-Outputimage */
*****/

for (il = 0L; il < total_bytes; il++)
{
    signal = (double) *(sig_ptr + skip_bytes + il);
    signal *= adjust;
    if (signal > 255.0) /* Check for overflow */
    {
        signal = 255.0;
    }
    *(cvbs_ptr + il) = (unsigned char) signal;
}

/*****
/* Get the Phase of the Color-Carrier (phase) */
*****/

phase = getburstphase (sig_ptr, carrier_fac, black_level, 5);
if (phase == NO_BURST)
{
    phase = getburstphase (sig_ptr, carrier_fac, black_level, 6);
}

```

```

        if (phase == NO_BURST)
        {
            fprintf (stderr, "No burst found in raw CVBS signal.\n");
            return (0);
        }
        fprintf (stdout, "First field in raw CVBS signal is field 1\n");
        if (skip_frames % 2)
        {
            fprintf (stdout, "First field in output CVBS signal is field 3\n");
        }
        else
        {
            fprintf (stdout, "First field in output CVBS signal is field 1\n");
        }
        phase -= 3.0 * M_PI / 4.0;                                /* -135 DEG */
    }
    else
    {
        fprintf (stdout, "First field in raw CVBS signal is field 3\n");
        if (skip_frames % 2)
        {
            fprintf (stdout, "First field in output CVBS signal is field 1\n");
        }
        else
        {
            fprintf (stdout, "First field in output CVBS signal is field 3\n");
        }
        phase -= 3.0 * M_PI / 4.0;                                /* -135 DEG */
    }

    omega = (double) (2.0 * M_PI / carrier_fac);

    /*****
    /* Create Color-Carrier-Outputimage */
    *****/

    for (il = 0L; il < total_bytes; il++)
    {
        signal = (double) sin (omega * (double) (il + skip_bytes) + phase);
        *(carrier_ptr + il) = (unsigned char) (signal * 127.0 + 128.0);
    }

    fprintf(stdout,"Done: Raw2CVBS\n");

return (1);
}

/*****
/*
/* Searches start of the first field (field 1 or field 3)
/*
/* Input:  Pointer (ptr) to signal-data
/*         Length (length) of signal in bytes
/*         Samplefrequency (fsc_fac * Frequency of Color-Carrier)
/*
/* Output: Offset to the first field
/*         Blacklevel (black_level)
/*
*****/

long  searchbegin (ptr, length, fsc_fac, black)
unsigned char  *ptr;
signed long    length;
double         fsc_fac;
unsigned char  *black;
{
double         val          = 0.0;
signed int     j            = 0,
count         = 0,
sync          = 0,

```

```

        numsync          = 0,
        thresh           = 3;
signed long   il         = 0L,
        pos             = 0L;
unsigned char c;
signed int    searchpattern [20]    = {481,86,481,86,481,86,481,86,481,86,
                                     37,529,37,529,37,529,37,529,37,529};

/*****
/* Adjust searchpattern on samplefrequency                               */
*****/

    thresh = thresh * (int) (fsc_fac / 4.0);
    for (j = 0L; j < 20; j++)
    {
        searchpattern[j] = searchpattern[j] * (int) (fsc_fac / 4.0);
    }

/*****
/* Search the pattern for v-sync (first field) in the Raw-CVBS-Signal    */
*****/

    while (il < length)
    {
        c = *(ptr + il);
        if (c > SYNCTHRESH)
        {
            if (sync == 0)
            {
                if (((searchpattern[numsync] - thresh) <= count) &&
                    ((searchpattern[numsync] + thresh) >= count))
                {
                    if (numsync == 0)
                    {
                        pos = il - count;
                    }
                    numsync++;
                }
                else
                {
                    numsync = 0;
                }
                sync = 1;
                count = 1;
            }
            else
            {
                count++;
            }
        }
        else
        {
            if (sync == 1)
            {
                if (((searchpattern[numsync] - thresh) <= count) &&
                    ((searchpattern[numsync] + thresh) >= count))
                {
                    numsync++;
                    if (numsync == 20)
                    {
                        for (j = count; j > 0; j--)
                        {
                            val += (double) *(ptr+il-j);
                        }
                        val /= (double) count;
                        *black = (unsigned char) ceil(val);
                        return (pos);
                    }
                }
            }
            else

```

```

        {
            numsync = 0;
        }
        sync = 0;
        count = 1;
    }
    else
    {
        count++;
    }
}

il++;
}

return (-1L);
}

/*****
/*
/* Gets the phase of the Color-Carrier (burst)
/*
/* Input:  Pointer (ptr) to CVBS-Signal
/*         Samplefrequency (fsc_fac * Frequency of Color-Carrier)
/*         Blacklevel (black_level)
/*         Number of the row to search burst for (row_number)
/*
/* Output: phase of the Color-Carrier at the beginning of the CVBS-Signal
/*
*****/

double getburstphase (ptr, fsc_fac, black_level, row_number)
unsigned char *ptr;
double fsc_fac;
unsigned char black_level;
signed int row_number;
{
    double phase = 0.0,
           ref_omega = 0.0,
           fsc_omega = 0.0,
           burst_amp = 0.0,
           err_val = 0.0,
           err_min = 9.9E99,
           a1, a2,
           pixel_row = 0.0,
           ref_sig [768],
           fsc_sig [13];
    signed long burst_offset = 0L,
               num_sin = 0L,
               i, j,
               step = 0L,
               pos = 0L;

    num_sin = (long) (4 * 8 * 12);
    fsc_omega = (double) (2.0 * M_PI / fsc_fac);
    ref_omega = (double) (2.0 * M_PI / (double) num_sin);
    pixel_row = (double) FCARRIER / FROW * fsc_fac;
    burst_amp = (double) black_level / 2.0;
    burst_offset = (long) (row_number * pixel_row) + (long) (25 * fsc_fac);
    step = (long) (num_sin / (long) fsc_fac);

    memset (&ref_sig[0], 0x00, 2 * num_sin * sizeof(double));

    for (i = 0; i < 2 * num_sin; i++) /* Reference-Sinus-Signal */
    {
        ref_sig[i] = burst_amp * sin(ref_omega * (double) i);
    }

    memset (&fsc_sig[0], 0x00, 13 * sizeof(double));
    for (i = 0; i <= (long) fsc_fac; i++) /* One period of burst */
    {

```

```

        fsc_sig[i] = (double) *(ptr+burst_offset+i) - (double) black_level;
    }

    for (j = 0; j < num_sin; j++)
    {
        err_val = 0.0;
        for (i = 0; i <= (long) fsc_fac; i++)
        {
            a1 = fsc_sig[i];
            a2 = ref_sig[j + i * step];
            err_val += fabs(a1 - a2);
        }
        if (err_val < err_min)
        {
            err_min = err_val;
            pos = j;
        }
    }

    if (err_min > 20.0 )
    {
        return (NO_BURST);
    }

    phase = ref_omega * (double) pos;
    phase -= fmod(fsc_omega * (double) (burst_offset), 2.0 * M_PI);

return (phase);
}
-library_code_end

-library_mods
-library_mods_end

```

A.2 cvbs2yc

A.2.1 cvbs2yc.pane

```

-F 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +0+0 'Standard-PAL Luminance and Chrominance Separation' cvbs2yc
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input CVBS Signal' 'Input CVBS Signal' i1
-I 1 0 1 0 0 0 50x1+2+4 +0+0 ' ' 'Input Impulse Response' 'Input Impulse Response' i2
-O 1 0 0 1 0 0 50x1+2+6 +0+0 ' ' 'Output Y Component' 'Output Y Component' o1
-O 1 0 0 1 0 0 50x1+2+7 +0+0 ' ' 'Output C Component' 'Output C Component' o2
-T 1 0 0 1 0 40x1+2+9 +0+0 3 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 0 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-T 1 0 0 1 0 40x1+2+12 +0+0 1 'Y/C Separation Method:' 'Y/C Separation Method' fm
-l 1 0 1 1 0 50x1+2+1 +0+0 0 'Simple Y/C Separation' 'False' 'True' 'Simple Y/C Separation' logical
-l 1 0 1 0 0 50x1+2+2 +0+0 0 'Adaptive Comb Filter' 'False' 'True' 'Adaptive Comb Filter.' logical
-l 1 0 1 0 0 50x1+2+3 +0+0 0 'Row Comb only' 'False' 'True' 'Row Comb only' logical
-l 1 0 1 0 0 50x1+2+4 +0+0 0 'Frame Comb only' 'False' 'True' 'Frame Comb only' logical
-l 1 0 1 0 0 50x1+2+5 +0+0 0 ' ' 'False' 'True' 'Variable (km)' logical
-E
-f 1 0 0 1 0 50x1+7+17 +0+0 0 1 0.5 'Variable (km)' 'Value for Filter-Mode (km)' km
-R 1 0 1 10x2+1+19 'Execute' 'do operation' $PALPLUS/bin/cvbs2yc
-Q 1 0 10x2+20+19 'Quit'
-H 1 10x2+39+19 'Help' 'man page for cvbs2yc' $PALPLUS/doc/manpages/cvbs2yc.1
-E
-E
-E

```

A.2.2 cvbs2yc.prog

```

-authors
Harald Zottmann
-authors_end

-short_prog_description
Filter which separates luminance and chrominance
from a CVBS signal (PAL).
-short_prog_description_end

-short_lib_description
Filter which separates luminance and chrominance
from a CVBS signal (PAL).
-short_lib_description_end

-man1_long_description
.I cvbs2yc
will separate luminance and chrominance from a
CVBS signal (PAL).
.LP
The CVBS signal generated from raw2cvbs will be
separated into its luminance component and its
chrominance component. The input impulse response
has to be a band-pass filter which is symmetrical
around the frequency of the color carrier (notch filter).
.LP
There are different filter modes selectable:
.LP
Simple y/c separation: notch filter for the chrominance
component (no comb filter).
.LP
Adaptive comb filter: adaptive soft switching between
a row comb filter and a frame comb filter. The switching
is performed by a motion detection in sequential fields.
.LP
Row comb only: Always row comb.
.LP
Frame comb only: Always frame comb.
.LP
Variable (km): km is used for the soft switching between the
two filter modes (row/frame comb). If km = 1, the filter mode
is row comb. If km = 0, the filter mode is frame comb. Every
value between 0 and 1 causes a mixing of the signals of both
filter modes.
.LP
(Teichner,D.: Adaptive Filter Techniques for
Separation of Luminance and Chrominance in PAL TV
Signals. IEEE Transactions on Consumer Electronics,
Vol. CE-32 (1986), No.3, pp 241-250)
-man1_long_description_end

-man1_examples
cvbs2yc -i1 cvbs.dat -i2 impulse_response
-o1 y_outimage -o2 c_outimage -f_sc 3 -fm 2
.LP
will generate y_outimage and c_outimage from cvbs.dat.
The Comb-Filter will work in the 'adaptive-mode'.
-man1_examples_end

-man1_restrictions
.I cvbs2yc
.LP
Datatypes of input and output signals:
.LP
input cvbs signal:      1_BYTE (0...255)
.LP
input impulse response: FLOAT
.LP

```

```

output signal y:      1_BYTE (16...235)
output signal c:      1_BYTE ( 0...255)
                      (128-127...128+127)

.LP
The length of the FIR-Filter must be odd-numbered.
-man1_restrictions_end

-man1_see_also
-man1_see_also_end

-man3_long_description
.I cvbs2yc
will separate luminance and chrominance from a
CVBS signal (PAL).
.LP
The CVBS signal generated from raw2cvbs will be
separated into its luminance component and its
chrominance component. The input impulse response
has to be a band-pass filter which is symmetrical
around the frequency of the color carrier (notch filter).
.LP
There are different filter modes selectable:
.LP
Simple y/c separation: notch filter for the chrominance
component (no comb filter).
.LP
Adaptive comb filter: adaptive soft switching between
a row comb filter and a frame comb filter. The switching
is performed by a motion detection in sequential fields.
.LP
Row comb only: Always row comb.
.LP
Frame comb only: Always frame comb.
.LP
Variable (km): km is used for the soft switching between the
two filter modes (row/frame comb). If km = 1, the filter mode
is row comb. If km = 0, the filter mode is frame comb. Every
value between 0 and 1 causes a mixing of the signals of both
filter modes.
-man3_long_description_end

-man3_restrictions
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

-usage_additions
-usage_additions_end

-include_includes
-include_includes_end

-include_additions
-include_additions_end

-include_macros
/*****
/* Read Input-Images */
*****/

#define READINPUT_CVBS(cvbs_img); \
    cvbs_img = readimage(cvbs2yc->i1_file);\
    if(cvbs_img == NULL)\
    {\
        (void) fprintf(stderr,"cvbs2yc : cannot read input image");\
        (void) fprintf(stderr," %s\n",cvbs2yc->i1_file);\
        exit(1);\
    }

```

```

#define READINPUT_FIRFIL(imp_res_img); \
    imp_res_img = readimage(cvbs2yc->i2_file);\
    if(imp_res_img == NULL)\
    {\
        (void) fprintf(stderr,"cvbs2yc : cannot read input image");\
        (void) fprintf(stderr," %s\n",cvbs2yc->i2_file);\
        exit(1);\
    }

/*****
/* Check Input-Images
*****/

#define CHECKINPUT_CVBS(cvbs_img); \
    (void) proptype(program, cvbs_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, cvbs_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, cvbs_img, 1, TRUE);\
    (void) proper_map_enable(program, cvbs_img, VFF_MAP_OPTIONAL, TRUE);

#define CHECKINPUT_FIRFIL(imp_res_img); \
    (void) proptype(program, imp_res_img, VFF_TYP_FLOAT, TRUE);\
    (void) proper_loc_type(program, imp_res_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, imp_res_img, 1, TRUE);\
    (void) proper_map_enable(program, imp_res_img, VFF_MAP_OPTIONAL, TRUE);
#include_macros_end

-main_variable_list
struct xvimage *cvbs_img,
               *imp_res_img,
               *y_comp_img,
               *c_comp_img,
               *readimage();
-main_variable_list_end

-main_before_lib_call
    if (check_args()) exit(1);

    READINPUT_CVBS (cvbs_img);
    CHECKINPUT_CVBS (cvbs_img);

    if (cvbs2yc->i2_flag == 1)
    {
        READINPUT_FIRFIL (imp_res_img);
        CHECKINPUT_FIRFIL (imp_res_img);
    }
-main_before_lib_call_end

-main_library_call
    if(! lcvbs2yc(cvbs_img, imp_res_img,
                 &y_comp_img, &c_comp_img,
                 cvbs2yc->f_sc_toggle, cvbs2yc->i2_flag,
                 cvbs2yc->km_float, cvbs2yc->fm_toggle) )
    {
        (void) fprintf(stderr, "lcvbs2yc failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if(! writeimage(cvbs2yc->o1_file, y_comp_img))
    {
        (void) fprintf(stderr,"cvbs2yc : unable to write y output file\n");
        exit(1);
    }
    if(! writeimage(cvbs2yc->o2_file, c_comp_img))
    {
        (void) fprintf(stderr,"cvbs2yc : unable to write c output file\n");
        exit(1);
    }
-main_after_lib_call_end

```

```

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lcvbs2yc(  cvbs_img,
               imp_res_img,
               y_comp_img,
               c_comp_img,
               f_sc_toggle,
               chroma_filter,
               km_var,
               fm_toggle)

struct  xvimage *cvbs_img,
          *imp_res_img,
          **y_comp_img,
          **c_comp_img;

int      f_sc_toggle,
          fm_toggle,
          chroma_filter;

float     km_var;
-library_def_end

-library_code
/*****
/*
/* Comb-Filter
/*
/*
/*****
/*****
/*
/* Library Code for cvbs2yc
/*
/*
/*****

#include "vinclude.h"

/*****

#define FCARRIER      4433618.75
#define FROW           15625

#define FC_FAC_4        1
#define FC_FAC_8        2
#define FC_FAC_12       3

#define YC_STANDARD     1
#define YC_ADAPTIVE     2
#define YC_ROW          3
#define YC_FRAME        4
#define YC_VARIABLE     5

/*****
/*
/* Main Function
/*
/*
/*****

{
int      convo ();

```

```

int          adaptive      = FALSE;
float        d             = 0.0,
            signal        = 0.0,
            carrier_fac   = 0.0,
            *fil_ptr      = NULL;
unsigned char *cvbs_ptr    = NULL,
            *y_ptr       = NULL,
            *c_ptr       = NULL;
signed char  *c_r_temp    = NULL,
            *c_f_temp    = NULL,
            *dm_temp     = NULL,
            *km_temp     = NULL,
            km;
signed long  il, kl,
            mean_hor     = 0L,
            mean_abs     = 0L,
            fil_length   = 0L,
            cvbs_length  = 0L,
            pixel_2rows  = 0L,
            pixel_4rows  = 0L,
            pixel_2frames = 0L,
            pixel_4frames = 0L;

/*****

fprintf(stdout, "\nCVBS2YC V1.1:\n");

switch (f_sc_toggle)
{
    case FC_FAC_4:
        carrier_fac = 4.0;
        break;

    case FC_FAC_8:
        carrier_fac = 8.0;
        break;

    case FC_FAC_12:
        carrier_fac = 12.0;
        break;

    default:
        fprintf (stderr, "Missing or improper argument in call to cvbs2yc.\n");
        return (0);
}

mean_hor = (long) (carrier_fac / 2.0);
mean_abs = (long) (carrier_fac + 3.0);

pixel_2rows = (long) (2.0 * FCARRIER / FROW * carrier_fac);      /* 2T_H */
pixel_4rows = (long) (4.0 * FCARRIER / FROW * carrier_fac);      /* 4T_H */

pixel_2frames = (long) (2.0 * FCARRIER / 25.0 * carrier_fac);    /* 2T_B */
pixel_4frames = (long) (4.0 * FCARRIER / 25.0 * carrier_fac);    /* 4T_B */

*****/

switch (fm_toggle)
{
    case YC_FRAME:
        km = 0;
        adaptive = FALSE;
        break;

    case YC_ADAPTIVE:
        km = 0;
        adaptive = TRUE;
        break;

    case YC_ROW:

```

```

        km = 127;
        adaptive = FALSE;
        break;

        case YC_VARIABLE:
        km = (signed char) (km_var * 127.0);
        adaptive = FALSE;
        break;

        case YC_STANDARD:
        break;

        default:
        fprintf (stderr, "Missing or improper argument in call to cvbs2yc.\n");
        break;
    }

/*****

    if (chroma_filter)
    {
        fil_length = imp_res_img->row_size;
        if (!(fil_length % 2L))
        {
            fprintf (stderr, "Filter-Length must be odd-numbered.\n");
            return (0);
        }
        fil_ptr = (float *) imp_res_img->imagedata;
    }

    cvbs_ptr    = (unsigned char *) cvbs_img->imagedata;
    cvbs_length = (signed long) (cvbs_img->row_size * cvbs_img->col_size);

/*****/
/*
/* Standard Y/C-Separation
/*
/*****/

/*****/
/* Allocate and set temporary memory for signal-processing
/*
/*****/

    if (fm_toggle == YC_STANDARD)
    {
        c_r_temp = (signed char *)
            malloc((unsigned) cvbs_length * sizeof(char));

        if (c_r_temp == NULL)
        {
            fprintf (stderr, "Out of memory.\n");
            return (0);
        }

        for (il = 0L; il < cvbs_length; il++)
        {
            *(c_r_temp + il) = (signed char) ((float) *(cvbs_ptr + il) - 128.0);
        }

/*****/
/* Allocate memory for Output-Images
/*
/*****/

        *y_comp_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
            1,1, "y_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
            VFF_LOC_IMPLICIT, 0);
        y_ptr = (unsigned char *) (*y_comp_img)->imagedata;

        *c_comp_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
            1,1, "c_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,

```

```

        VFF_LOC_IMPLICIT, 0);
c_ptr = (unsigned char *) (*c_comp_img)->imagedata;

if ((*y_comp_img == NULL) || (*c_comp_img == NULL))
{
    if (*y_comp_img != NULL) freeimage(*y_comp_img);
    if (*c_comp_img != NULL) freeimage(*c_comp_img);
    if (c_r_temp != NULL) free(c_r_temp);
    fprintf (stderr, "Out of memory.\n");
    return (0);
}

/*****
/* Copy CVBS-Signal to Y-Component */
*****/

for (il = 0L; il < cvbs_length; il++)
{
    *(y_ptr + il) = *(cvbs_ptr + il);
}

/*****
/* Convolution with Impulse-Response */
*****/

if (chroma_filter)
{
    if (!convo (c_r_temp, fil_ptr, cvbs_length, fil_length))
    {
        fprintf (stderr, "Convolution failed.\n");
        return (0);
    }
}

/*****
/* Subtract C-Component from CVBS-Signal to get Y-Component */
*****/

if (chroma_filter)
{
    for (il = 0L; il < cvbs_length; il++)
    {
        signal = ((float) *(y_ptr + il) - (float) *(c_r_temp + il));
        if (signal > 255.0) signal = 255.0;
        if (signal < 0.0) signal = 0.0;
        *(y_ptr + il) = (unsigned char) signal;
        *(c_ptr + il) = (unsigned char) ((float) *(c_r_temp + il) + 128.0);
    }
    free (c_r_temp);
} /* fm_toggle == YC_STANDARD */

/*****
/*
/* Comb-Filter */
/*
*****/

/*****
/* Allocate and set temporary memory for signal-processing */
*****/

else
{
    dm_temp = (signed char *)
        malloc((unsigned) (cvbs_length + pixel_4frames) * sizeof(char));

    km_temp = (signed char *)
        malloc((unsigned) (cvbs_length + pixel_4frames) * sizeof(char));

```

```

if ((dm_temp == NULL) || (km_temp == NULL))
{
    if (dm_temp != NULL) free(dm_temp);
    if (km_temp != NULL) free(km_temp);
    fprintf (stderr, "Out of memory.\n");
    return (0);
}

for (il = 0L; il < cvbs_length + pixel_4frames; il++)
{
    *(dm_temp + il) = 0;
    *(km_temp + il) = 0;
}

/*****
/* Motion-Detection-Signal
*****/

if (adaptive == TRUE) /* Adaptive-Mode */
{
    for (il = 0L; il < cvbs_length; il++)
    {
        *(dm_temp + il)
            += (signed char) (0.50 * (float) *(cvbs_ptr + il));
        *(dm_temp + il + pixel_4frames)
            -= (signed char) (0.50 * (float) *(cvbs_ptr + il));
    }

    /* Rectifier */

    for (il = pixel_4frames; il < cvbs_length; il++)
    {
        *(dm_temp + il) = (signed char) abs((int) *(dm_temp + il));
    }

    /* X-Y-Averaging */

    for (il = pixel_4frames; il < cvbs_length; il++)
    {
        d = (float) *(dm_temp + il);
        d += (float) *(dm_temp + il - pixel_2rows);
        d += (float) *(dm_temp + il + pixel_2rows);
        for (kl = 1L; kl <= mean_hor; kl++)
        {
            d += (float) *(dm_temp + il - kl);
            d += (float) *(dm_temp + il + kl);
        }
        *(km_temp + il) = (signed char) (d / (float) mean_abs);
    }

    /* Non-Linear Transfer-Function */

    for (il = pixel_4frames; il < cvbs_length; il++)
    {
        d = (float) *(km_temp + il) / 127.0;
        if (d > 0.10)
        {
            d = 1.0;
        }
        else
        {
            d = 100.0 * ((3.0 * d * d) - (20.0 * d * d * d));
        }
        *(km_temp + il) = (signed char) (d * 127.0);
    }

    /* Row-Comb for first-two and last-two frames */

    for (il = 0L; il < pixel_2frames; il++)
    {
        *(km_temp + pixel_2frames + il) = (signed char) 127;
        *(km_temp + cvbs_length + il) = (signed char) 127;
    }
}

```

```

else                                     /* Non-Adaptive-Mode */
{
    for (il = pixel_2frames; il < cvbs_length + pixel_2frames; il++)
    {
        *(km_temp + il) = km;
    }
}

free (dm_temp);

/*****
/* Allocate and set temporary memory for signal-processing */
*****/

c_r_temp = (signed char *)
            malloc((unsigned) (cvbs_length + pixel_4frames) * sizeof(char));

c_f_temp = (signed char *)
            malloc((unsigned) (cvbs_length + pixel_4frames) * sizeof(char));

if ((c_r_temp == NULL) || (c_f_temp == NULL))
{
    if (c_r_temp != NULL) free(c_r_temp);
    if (c_f_temp != NULL) free(c_f_temp);
    if (km_temp != NULL) free(km_temp);
    fprintf (stderr, "Out of memory.\n");
    return (0);
}

for (il = 0L; il < cvbs_length + pixel_4frames; il++)
{
    *(c_r_temp + il) = 0;
    *(c_f_temp + il) = 0;
}

/*****
/* Frame-Comb-Signal */
*****/

for (il = 0L; il < cvbs_length; il++)          /* FCF */
{
    *(c_f_temp + il)
        -= (signed char) (0.25 * (float) *(cvbs_ptr + il));
    *(c_f_temp + il + pixel_2frames)
        += (signed char) (0.50 * (float) *(cvbs_ptr + il));
    *(c_f_temp + il + pixel_4frames)
        -= (signed char) (0.25 * (float) *(cvbs_ptr + il));
}

/*****
/* Row-Comb-Signal */
*****/

for (il = 0L; il < cvbs_length; il++)          /* RCF */
{
    *(c_r_temp + il + pixel_2frames - pixel_2rows)
        -= (signed char) (0.25 * (float) *(cvbs_ptr + il));
    *(c_r_temp + il + pixel_2frames)
        += (signed char) (0.50 * (float) *(cvbs_ptr + il));
    *(c_r_temp + il + pixel_2frames + pixel_2rows)
        -= (signed char) (0.25 * (float) *(cvbs_ptr + il));
}

/*****
/* Switch between Filter-Modes */
*****/

for (il = pixel_2frames; il < cvbs_length + pixel_2frames; il++)
{
    *(c_f_temp + il) += (signed char)

```

```

        ((float) *(km_temp + il) / 127.0)
        * ((float) *(c_r_temp + il) - (float) *(c_f_temp + il)) );
    }

    free (km_temp);

/*****
/* Shift back Color-Component
*****/

    for (il = 0L; il < cvbs_length; il++)
    {
        *(c_r_temp + il) = *(c_f_temp + il + pixel_2frames);
    }

    free (c_f_temp);

/*****
/* Convolution with Impulse-Response
*****/

    if (chroma_filter)
    {
        if (!convo (c_r_temp, fil_ptr, cvbs_length, fil_length))
        {
            fprintf (stderr, "Convolution failed.\n");
            free (c_r_temp);
            return (0);
        }
    }

/*****
/* Allocate memory for Output-Images
*****/

    *y_comp_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
                               1,1, "y_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
                               VFF_LOC_IMPLICIT, 0);
    y_ptr = (unsigned char *) (*y_comp_img)->imagedata;

    *c_comp_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
                               1,1, "c_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
                               VFF_LOC_IMPLICIT, 0);
    c_ptr = (unsigned char *) (*c_comp_img)->imagedata;

    if ((*y_comp_img == NULL) || (*c_comp_img == NULL))
    {
        if (*y_comp_img != NULL) freeimage(*y_comp_img);
        if (*c_comp_img != NULL) freeimage(*c_comp_img);
        if (c_r_temp != NULL) free(c_r_temp);
        fprintf (stderr, "Out of memory.\n");
        return (0);
    }

/*****
/* Copy CVBS-Signal to Y-Component
*****/

    for (il = 0L; il < cvbs_length; il++)
    {
        *(y_ptr + il) = *(cvbs_ptr + il);
    }

/*****
/* Subtract C-Component from CVBS-Signal to get Y-Component
*****/

    for (il = 0L; il < cvbs_length; il++)
    {
        *(y_ptr + il) = (unsigned char)

```

```

        ((float) *(y_ptr + il) - (float) *(c_r_temp + il));
        *(c_ptr + il) = (unsigned char)
            ((float) *(c_r_temp + il) + 128.0);
    }

    free (c_r_temp);
} /* fm_toggle == YC_ROW || YC_FRAME || YC_ADAPTIVE || YC_VARIABLE */

/*****
/*
/* End of Standard Y/C-Separation and of Comb-Filter
/*
/*
*****/

/*****
/* Scale Y-Component according to CCIR-Rec.601
*****/

    for (il = 0L; il < cvbs_length; il++)
    {
        signal = (float) *(y_ptr + il);
        signal = (signal / 143.0 * 219.0) - 76.0;
        if (signal > 235.0) signal = 235.0;
        if (signal < 0.0) signal = 0.0;
        *(y_ptr + il) = (unsigned char) signal;
    }

    fprintf(stdout,"Done: CVBS2YC\n");

return (1);
}

/*****
/*
/* Performs a circular convolution of the Color-Component (signed char)
/* with the Impulse-Response (float).
/*
/*
/* Input:  Data vector of Color-Component (datain1)
/*         Data vector of Impulse-Response (datain2)
/*         Number of data points (dpoints1, dpoints2)
/*
/* Output: Resultant convolution data (datain1)
/*
*****/

int    convo (datain1, datain2, dpoints1, dpoints2)
signed long    dpoints1,
               dpoints2;
signed char    *datain1;
float          *datain2;
{
    signed long    shift,
                  idx,
                  n, h;
    float          *conv,
                  temp;

    if (dpoints1 < dpoints2)
    {
        fprintf(stderr,"Data vector 1 must be longer than Data vector 2\n");
        return (0);
    }

    conv = (float *) malloc((unsigned)(dpoints1 * sizeof(float)));
    if (conv == NULL)
    {
        fprintf(stderr,"Out of memory.\n");
        return (0);
    }

```

```

    for (n = 0L; n < dpoints1; n++)
    {
        temp = 0.0;
        for (h = 0L; h < dpoints2; h++)
        {
            idx = n - h;
            if (idx < 0L)
            {
                idx += dpoints1;
            }
            temp += datain2[h] * (float) datain1[idx];
        }
        conv[n] = temp;
    }

    for (n = 0L; n < dpoints1; n++)
    {
        temp = conv[n];
        if (temp > 127.0)
        {
            temp = 127.0;
        }
        if (temp < -126.0)
        {
            temp = -126.0;
        }
        datain1[n] = (signed char) temp;
    }

    shift = dpoints2 / 2;
    for (n = 0L; n < dpoints1 - shift; n++)
    {
        datain1[n] = datain1[n + shift];
    }

    free (conv);

return (1);
}
-library_code_end

-library_mods
-library_mods_end

```

A.3 c2uv

A.3.1 c2uv.pane

```

-F 4.2 1 0 170x 7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +2+0 'U/V-Demodulator' c2uv
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input C Signal' 'Input C Signal' i1
-I 1 0 0 1 0 0 50x1+2+4 +0+0 ' ' 'Input Color Carrier' 'Input Color Carrier' i2
-I 1 0 1 0 0 0 50x1+2+6 +0+0 ' ' 'Input Impulse Response' 'Input Impulse Response' i3
-O 1 0 0 1 0 0 50x1+2+8 +0+0 ' ' 'Output U Component' 'Output U Component' o1
-O 1 0 0 1 0 0 50x1+2+9 +0+0 ' ' 'Output V Component' 'Output V Component' o2
-T 1 0 0 1 0 40x1+2+11 +0+0 2 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 1 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-T 1 0 0 1 0 40x1+2+14 +0+0 1 'PAL Delay-Line:' 'PAL Delay-Line' dl
-l 1 0 1 1 0 1x1+2+1 +0+0 0 'Use PAL Delay-Line' 'False' 'True' 'Use PAL Delay-Line' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 'No PAL Delay-Line' 'False' 'True' 'No PAL Delay-Line' logical
-E
-H 1 10x2+39+17 'Help' 'man page for c2uv' $PALPLUS/doc/manpages/c2uv.1

```

```
-Q 1 0 10x2+20+17 'Quit'
-R 1 0 1 10x2+1+17 'Execute' 'do operation' $PALPLUS/bin/c2uv
-E
-E
-E
```

A.3.2 c2uv.prog

```
-authors
Harald Zottmann
-authors_end

-short_prog_description
Demodulates the u/v component from the c component.
-short_prog_description_end

-short_lib_description
Demodulates the u/v component from the c component.
-short_lib_description_end

-man1_long_description
.I c2uv
will demodulate the u/v component from the c component
generated by cvbs2yc. The second input requires the
color carrier which is provided by raw2cvbs.
.LP
The u/v Component will be demodulated from the c component
by using the color carrier, which is refreshed from a
raw CVBS signal by raw2cvbs. The input for the impulse
response has to be connected with a low-pass filter for the
demodulated u/v component. The use of the PAL delay-line is
optional.
-man1_long_description_end

-man1_examples
c2uv -i1 c_component -i2 color_carrier -i3 impulse_response
-o1 u_outimage -o2 v_outimage -f_sc 2 -dl 1
.LP
will generate u_outimage and v_outimage from
c_component and color_carrier. The PAL delay-line
is activated and the samplefrequency is set to the
carrier-frequency multiplied with four. After demodulation,
the u/v component signal will be filtered using the
impulse_response.
-man1_examples_end

-man1_restrictions
.I c2uv
.LP
Datatypes of input and output signals:
.LP
input c component:      1_BYTE (0...255)
                        (128-127...128+127)
input color carrier:    1_BYTE (0...255)
                        (128-127...128+127)
.LP
input impulse response: FLOAT
.LP
output signal u:        1_BYTE (16...240)
                        (128-112...128+112)
output signal v:        1_BYTE (16...240)
                        (128-112...128+112)
-man1_restrictions_end

-man1_see_also
-man1_see_also_end

-man3_long_description
```

```

.I c2uv
will demodulate the u/v component from the c component
generated by cvbs2yc. The second input requires the
color carrier which is provided by raw2cvbs.
.LP
The u/v Component will be demodulated from the c component
by using the color carrier, which is refreshed from a
raw CVBS signal by raw2cvbs. The input for the impulse
response has to be connected with a low-pass filter for the
demodulated u/v component. The use of the PAL delay-line is
optional.
-man3_long_description_end

-man3_restrictions
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

-usage_additions
-usage_additions_end

-include_includes
-include_includes_end

-include_additions
-include_additions_end

-include_macros
/*****
/* Read Input-Images */
*****/

#define READINPUT(c_comp_img, carrier_img); \
    c_comp_img = readimage(c2uv->i1_file);\
    if(c_comp_img == NULL)\
    {\
        (void) fprintf(stderr,"c2uv : cannot read input image");\
        (void) fprintf(stderr," %s\n",c2uv->i1_file);\
        exit(1);\
    }\
    carrier_img = readimage(c2uv->i2_file);\
    if(carrier_img == NULL)\
    {\
        (void) fprintf(stderr,"c2uv : cannot read input image");\
        (void) fprintf(stderr," %s\n",c2uv->i2_file);\
        exit(1);\
    }\

#define READINPUT_FIRFIL(imp_res_img); \
    imp_res_img = readimage(c2uv->i3_file);\
    if(imp_res_img == NULL)\
    {\
        (void) fprintf(stderr,"c2uv : cannot read input image");\
        (void) fprintf(stderr," %s\n",c2uv->i3_file);\
        exit(1);\
    }\

/*****
/* Check Input-Images */
*****/

#define CHECKINPUT(c_comp_img, carrier_img); \
    (void) proptype(program, c_comp_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proptype(program, carrier_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, c_comp_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_loc_type(program, carrier_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, c_comp_img, 1, TRUE);\
    (void) proper_num_bands(program, carrier_img, 1, TRUE);\
    (void) proper_map_enable(program, c_comp_img, VFF_MAP_OPTIONAL, TRUE);\

```

```

        (void) proper_map_enable(program, carrier_img, VFF_MAP_OPTIONAL, TRUE);\
        (void) matchsize(program, c_comp_img, carrier_img, TRUE);

#define CHECKINPUT_FIRFIL(imp_res_img); \
        (void) proptype(program, imp_res_img, VFF_TYP_FLOAT, TRUE);\
        (void) proper_loc_type(program, imp_res_img, VFF_LOC_IMPLICIT, TRUE);\
        (void) proper_num_bands(program, imp_res_img, 1, TRUE);\
        (void) proper_map_enable(program, imp_res_img, VFF_MAP_OPTIONAL, TRUE);\
#include_macros_end

-main_variable_list
struct xvimage *c_comp_img,
               *carrier_img,
               *imp_res_img,
               *u_comp_img,
               *v_comp_img,
               *readimage();
-main_variable_list_end

-main_before_lib_call
    if(check_args()) exit(1);

    READINPUT (c_comp_img, carrier_img);
    CHECKINPUT (c_comp_img, carrier_img);

    if (c2uv->i3_flag == 1)
    {
        READINPUT_FIRFIL (imp_res_img);
        CHECKINPUT_FIRFIL (imp_res_img);
    }
-main_before_lib_call_end

-main_library_call
    if(! lc2uv(c_comp_img, carrier_img, imp_res_img,
              &u_comp_img, &v_comp_img,
              c2uv->i3_flag,
              c2uv->f_sc_toggle, c2uv->dl_toggle) )
    {
        fprintf(stderr, "lc2uv failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if(! writeimage(c2uv->o1_file, u_comp_img))
    {
        (void) fprintf(stderr, "c2uv : unable to write u output file\n");
        exit(1);
    }
    if(! writeimage(c2uv->o2_file, v_comp_img))
    {
        (void) fprintf(stderr, "c2uv : unable to write v output file\n");
        exit(1);
    }
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lc2uv(
    c_comp_img,
    carrier_img,
    imp_res_img,
    u_comp_img,

```

```

        v_comp_img,
        lowpass_filter,
        f_sc_toggle,
        dl_toggle)

struct ximage *c_comp_img,
               *carrier_img,
               *imp_res_img,
               **u_comp_img,
               **v_comp_img;

int           lowpass_filter,
               f_sc_toggle,
               dl_toggle;
-library_def_end

-library_code
/*****
/*
/* Library Code for c2uv
/*
/*
*****/

#include "vinclude.h"

/*****

#define FCARRIER      4433618.75
#define FROW           15625

#define FC_FAC_4       1
#define FC_FAC_8       2
#define FC_FAC_12      3

#define DL_USE         1
#define DL_NO          2

/*****
/*
/* Main Function
/*
/*
*****/
{
int           convo      ();
int           field3     ();

double        carrier_fac = 0.0,
               signal     = 0.0,
               delaysignal = 0.0,
               f_pixel_row = 0.0;
float         *fil_ptr    = NULL;
unsigned char *c_ptr      = NULL,
               *carrier_ptr = NULL,
               *u_ptr      = NULL,
               *v_ptr      = NULL;
unsigned long  il         = 0L,
               fil_length = 0L,
               burstmode  = 0L,
               deg_90     = 0L,
               deg_180    = 0L,
               l_pixel_row = 0L,
               c_length   = 0L;

/*****

fprintf(stdout, "\nC2UV V1.3:\n");

switch (f_sc_toggle)
{
    case FC_FAC_4:

```

```

        carrier_fac = 4.0;
        break;

        case FC_FAC_8:
        carrier_fac = 8.0;
        break;

        case FC_FAC_12:
        carrier_fac = 12.0;
        break;

        default:
        fprintf (stderr, "Missing or improper argument in call to c2uv.\n");
        return(0);
    }

/*****

    if (lowpass_filter)
    {
        fil_length = imp_res_img->row_size;
        if (!(fil_length % 2L))
        {
            fprintf (stderr, "Filter-Length must be odd-numbered.\n");
            return (0);
        }
        fil_ptr = (float *) imp_res_img->imagedata;
    }

/*****

    c_length    = (unsigned long) (c_comp_img->col_size * c_comp_img->row_size);
    c_ptr       = (unsigned char *) c_comp_img->imagedata;
    carrier_ptr = (unsigned char *) carrier_img->imagedata;

/*****

    *u_comp_img = createimage ((unsigned long) (c_length), 1L, VFF_TYP_1_BYTE,
                               1,1, "u_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
                               VFF_LOC_IMPLICIT, 0);

    if (*u_comp_img == NULL)
    {
        fprintf (stderr, "Out of memory.\n");
        return(0);
    }

    u_ptr = (unsigned char *) (*u_comp_img)->imagedata;

/*****

    *v_comp_img = createimage ((unsigned long) (c_length), 1L, VFF_TYP_1_BYTE,
                               1,1, "v_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
                               VFF_LOC_IMPLICIT, 0);

    if (*v_comp_img == NULL)
    {
        fprintf (stderr, "Out of memory.\n");
        return(0);
    }

    v_ptr = (unsigned char *) (*v_comp_img)->imagedata;

/*****
/* f_pixel_row = Number of pixels (double) */
/* l_pixel_row = Number of pixels (long) for PAL-Delay-Line (64.057 us) */
/* 64.057 us = 64.000 us + 57.000 ns (284 Periods of Color-Carrier) */
/*****

    f_pixel_row    = (double) (FCARRIER / FROW * carrier_fac);

```

```

        l_pixel_row    = (long) f_pixel_row + (long) (0.25 * carrier_fac);

/*****
/* Demodulate U-Component
/* Create U-Component-Outputimage
*****/

    if (field3 (c_ptr, carrier_fac) == 1)
    {
        fprintf (stdout, "Starting with field 3\n");
        burstmode = 1L;
    }
    else
    {
        fprintf (stdout, "Starting with field 1\n");
        burstmode = 0L;
    }

/*****
/* No PAL-Delay-Line (U-Component)
*****/

    if (dl_toggle == DL_NO)
    {
        for (il = 0L; il < c_length; il++)
        {
            signal = (double) *(c_ptr + il) - 128.0;
            signal *= (double) *(carrier_ptr + il) - 128.0;
            *(u_ptr + il) = (unsigned char) (signal / 128.0 + 128.0);
        }

/*****
/* Use PAL-Delay-Line (U-Component)
*****/

    else
    {
        for (il = 0L; il < c_length; il++)
        {
            signal = (double) *(c_ptr + il) - 128.0;
            if (il > l_pixel_row)
            {
                delaysignal = (double) *(c_ptr + il - l_pixel_row) - 128.0;
            }
            else
            {
                delaysignal = 0.0;
            }
            signal += delaysignal;
            signal *= (double) *(carrier_ptr + il) - 128.0;
            *(u_ptr + il) = (unsigned char) (signal / 255.0 + 128.0);
        }

/*****
/* Demodulate V-Component
/* Create V-Component-Outputimage
*****/

    deg_90 = (long) (carrier_fac / 4.0);
    deg_180 = (long) (carrier_fac / 2.0);
    carrier_ptr += deg_90;                                     /* +90 DEG */

/*****
/* No PAL-Delay-Line (V-Component)
*****/

    if (dl_toggle == DL_NO)
    {

```

```

        for (il = 0L; il < c_length; il++)
        {
            if ((long) ((double) il / f_pixel_row) > burstmode)
            {
                burstmode++;
                if (burstmode % 2L)
                {
                    carrier_ptr -= deg_180;
                }
            }
            else
            {
                carrier_ptr += deg_180;
            }
        }
        signal = (double) *(c_ptr + il) - 128.0;
        signal *= (double) *(carrier_ptr + il) - 128.0;
        *(v_ptr + il) = (unsigned char) (signal / 128.0 + 128.0);
    }

    /*****
    /* Use PAL-Delay-Line (V-Component) */
    /*****/

    else
    {
        for (il = 0L; il < c_length; il++)
        {
            signal = (double) *(c_ptr + il) - 128.0;
            if (il > l_pixel_row)
            {
                delaysignal = (double) *(c_ptr + il - l_pixel_row) - 128.0;
            }
            else
            {
                delaysignal = 0.0;
            }
            if ((long) ((double) il / f_pixel_row) > burstmode)
            {
                burstmode++;
                if (burstmode % 2L)
                {
                    carrier_ptr -= deg_180;
                }
            }
            else
            {
                carrier_ptr += deg_180;
            }
        }
        signal -= delaysignal;
        signal *= (double) *(carrier_ptr + il) - 128.0;
        *(v_ptr + il) = (unsigned char) (signal / 255.0 + 128.0);
    }

    /*****
    /* Convolution with Impulse-Response */
    /*****/

    if (lowpass_filter)
    {
        if (!convo (u_ptr, fil_ptr, c_length, fil_length))
        {
            fprintf (stderr, "Convolution (U-Component) failed.\n");
            return (0);
        }
        if (!convo (v_ptr, fil_ptr, c_length, fil_length))
        {
            fprintf (stderr, "Convolution (V-Component) failed.\n");
            return (0);
        }
    }

```

```

    }
}

/*****
/* Scale U/V-Components according to CCIR-Rec.601 */
*****/

for (il = 0L; il < c_length; il++)
{
    signal = (double) *(u_ptr + il);
    signal = (signal - 128.0) / 30.0;
    if (signal > 1.0) signal = 1.0;
    if (signal < -1.0) signal = -1.0;
    *(u_ptr + il) = (unsigned char) (signal * 112.0) + 128.0;

    signal = (double) *(v_ptr + il);
    signal = (signal - 128.0) / 42.0;
    if (signal > 1.0) signal = 1.0;
    if (signal < -1.0) signal = -1.0;
    *(v_ptr + il) = (unsigned char) (signal * 112.0) + 128.0;
}

fprintf(stdout, "Done: C2UV\n");

return (1);
}

/*****
/*
/* Performs a circular convolution of 1_BYTE-Signals (unsigned char)
/* with an Impulse-Response (float).
/*
/*
/* Input:  Data vector of Color-Component      (datain1)
/*          Data vector of Impulse-Response    (datain2)
/*          Number of data points              (dpoints1, dpoints2)
/*
/* Output: Resultant convolution data          (datain1)
/*
*****/

int    convo (datain1, datain2, dpoints1, dpoints2)
signed long    dpoints1,
               dpoints2;
unsigned char  *datain1;
float          *datain2;
{
    signed long    shift,
                  idx,
                  n, h;
    float          *conv,
                  temp;

    if (dpoints1 < dpoints2)
    {
        fprintf(stderr, "Data vector 1 must be longer than Data vector 2\n");
        return (0);
    }

    conv = (float *) malloc((unsigned)(dpoints1 * sizeof(float)));
    if (conv == NULL)
    {
        fprintf(stderr, "Out of memory.\n");
        return (0);
    }

    for (n = 0L; n < dpoints1; n++)
    {
        temp = 0.0;
        for (h = 0L; h < dpoints2; h++)
        {

```

```

        idx = n - h;
        if (idx < 0L)
        {
            idx += dpoints1;
        }
        temp += datain2[h] * (float) datain1[idx];
    }
    conv[n] = temp;
}

for (n = 0L; n < dpoints1; n++)
{
    temp = conv[n];
    if (temp > 255.0)
    {
        temp = 255.0;
    }
    if (temp < 0.0)
    {
        temp = 0.0;
    }
    datain1[n] = (unsigned char) temp;
}

shift = dpoints2 / 2;
for (n = 0L; n < dpoints1 - shift; n++)
{
    datain1[n] = datain1[n + shift];
}

free (conv);

return (1);
}

/*****
/*
/* Checks for burst in field 3
/*
/* Input: Pointer (ptr) to Color-Component
/* Samplefrequency (fsc_fac * Frequency of Color-Carrier)
/*
/* Returncode: 1 if burst found (--> field 3) - 0 if burst not found (--> field 1)
/*
*****/

int field3 (ptr, fsc_fac)
unsigned char *ptr;
double fsc_fac;
{
    signed long burst_6_offset = 0L,
                burst_9_offset = 0L,
                burst_length = 0L,
                mean_start = 0L,
                mean_end = 0L,
                il;
    float pixel_row = 0.0,
           mean_level = 0.0,
           val = 0.0,
           sum_6 = 0.0,
           sum_9 = 0.0;

    pixel_row = (float) (FCARRIER / FROW * fsc_fac);
    burst_6_offset = (signed long) ( 5.0 * pixel_row) + (long) ( 25 * fsc_fac);
    burst_9_offset = (signed long) ( 8.0 * pixel_row) + (long) ( 25 * fsc_fac);
    mean_start = (signed long) ( 5.0 * pixel_row) + (long) ( 50 * fsc_fac);
    mean_end = (signed long) ( 5.0 * pixel_row) + (long) (250 * fsc_fac);
    burst_length = (signed long) ( 8.0 * fsc_fac);

    for (il = mean_start; il < mean_end; il++)

```

```

        {
            mean_level += (float) *(ptr + il);
        }
    mean_level /= (float) (mean_end - mean_start);

    for (il = 0L; il < burst_length; il++) /* Burstamplitude row 6 is ??? % */
    {
        val = (float) *(ptr + burst_6_offset + il);
        val -= (float) mean_level;
        sum_6 += (float) abs ((int) val);
    }

    for (il = 0L; il < burst_length; il++) /* Burstamplitude row 9 is 100 % */
    {
        val = (float) *(ptr + burst_9_offset + il);
        val -= (float) mean_level;
        sum_9 += (float) abs ((int) val);
    }

    if (sum_9 > 0.0)
    {
        val = sum_6 / sum_9;
    }
    else
    {
        return (-1); /* Error */
    }

    if (val > 0.375) /* Threshold between 0.25 and 0.50 */
    {
        return (1); /* Field 3 */
    }

    return (0); /* Field 1 */
}
-library_code_end

-library_mods
-library_mods_end

```

A.4 yuv2rgb

A.4.1 yuv2rgb.pane

```

-F 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +2+0 'PAL De-Matrix' yuv2rgb
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input Y Component' 'y input image' iy
-I 1 0 0 1 0 0 50x1+2+3 +0+0 ' ' 'Input U Component' 'u input image' iu
-I 1 0 0 1 0 0 50x1+2+4 +0+0 ' ' 'Input V Component' 'v input image' iv
-O 1 0 0 1 0 0 50x1+2+6 +0+0 ' ' 'Output R Component' 'r output image' or
-O 1 0 0 1 0 0 50x1+2+7 +0+0 ' ' 'Output G Component' 'g output image' og
-O 1 0 0 1 0 0 50x1+2+8 +0+0 ' ' 'Output B Component' 'b output image' ob
-R 1 0 1 10x2+1+10 'Execute' 'execute yuv2rgb' $PALPLUS/bin/yuv2rgb
-Q 1 0 10x2+20+10 'Quit'
-H 1 10x2+39+10 'Help' 'man page for yuv2rgb' $PALPLUS/doc/manpages/yuv2rgb.1
-E
-E
-E

```

A.4.2 yuv2rgb.prog

```

-authors
Harald Zottmann
-authors_end

```

```

-short_prog_description
Transforms YUV to RGB.
-short_prog_description_end

-short_lib_description
Transforms YUV to RGB.
-short_lib_description_end

-man1_long_description
.I yuv2rgb
transforms YUV to RGB, using the matrix:
.LP
  R = 1.000 Y + 0.000 U + 1.140 V
  G = 1.000 Y - 0.394 U - 0.581 V
  B = 0.998 Y + 2.028 U + 0.000 V
.LP
The input data must be of the following type:
.LP
  Y 16...235
  U 16...240 (128-112...128+112)
  V 16...240 (128-112...128+112)
.LP
  Y BLACK: 16
  Y WHITE: 235
  U/V NO COLOR: 128
-man1_long_description_end

-man1_examples
-man1_examples_end

-man1_restrictions
.I lyuv2rgb
works only with VFF_TYP_1_BYTE.
All input images must be of the same size.
-man1_restrictions_end

-man1_see_also
-man1_see_also_end

-man3_long_description
.I yuv2rgb
transforms YUV to RGB, using the matrix:
.LP
  R = 1.000 Y + 0.000 U + 1.140 V
  G = 1.000 Y - 0.394 U - 0.581 V
  B = 0.998 Y + 2.028 U + 0.000 V
.LP
The input data must be of the following type:
.LP
  Y 16...235
  U 16...240 (128-112...128+112)
  V 16...240 (128-112...128+112)
.LP
  Y BLACK: 16
  Y WHITE: 235
  U/V NO COLOR: 128
-man3_long_description_end

-man3_restrictions
.I lyuv2rgb
VFF_TYP_1_BYTE required.
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

-usage_additions
-usage_additions_end

```

```

#include_includes
#include_includes_end

#include_additions
#include_additions_end

#include_macros
/*****
/* Read Input-Images (YUV) */
*****/

#define READINPUT(image_in_y, image_in_u, image_in_v); \
    image_in_y = readimage(yuv2rgb->iy_file);\
    if(image_in_y == NULL)\
    {\
        (void) fprintf(stderr,"yuv2rgb : cannot read input image");\
        (void) fprintf(stderr," %s\n",yuv2rgb->iy_file);\
        exit(1);\
    }\
    image_in_u = readimage(yuv2rgb->iu_file);\
    if(image_in_u == NULL)\
    {\
        (void) fprintf(stderr,"yuv2rgb : cannot read input image");\
        (void) fprintf(stderr," %s\n",yuv2rgb->iu_file);\
        exit(1);\
    }\
    image_in_v = readimage(yuv2rgb->iv_file);\
    if(image_in_v == NULL)\
    {\
        (void) fprintf(stderr,"yuv2rgb : cannot read input image");\
        (void) fprintf(stderr," %s\n",yuv2rgb->iv_file);\
        exit(1);\
    }\

/*****
/* Check Input-Images (YUV) */
*****/

#define CHECKINPUT(image_in_y, image_in_u, image_in_v); \
    (void) proptype(program, image_in_y, VFF_TYP_1_BYTE, TRUE);\
    (void) proptype(program, image_in_u, VFF_TYP_1_BYTE, TRUE);\
    (void) proptype(program, image_in_v, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, image_in_y, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_loc_type(program, image_in_u, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_loc_type(program, image_in_v, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, image_in_y, 1, TRUE);\
    (void) proper_num_bands(program, image_in_u, 1, TRUE);\
    (void) proper_num_bands(program, image_in_v, 1, TRUE);\
    (void) proper_map_enable(program, image_in_y, VFF_MAP_OPTIONAL, TRUE);\
    (void) proper_map_enable(program, image_in_u, VFF_MAP_OPTIONAL, TRUE);\
    (void) proper_map_enable(program, image_in_v, VFF_MAP_OPTIONAL, TRUE);\
    (void) matchsize(program, image_in_y, image_in_u, TRUE);\
    (void) matchsize(program, image_in_y, image_in_v, TRUE);

#include_macros_end

-main_variable_list
struct xvimage *image_in_y,
               *image_in_u,
               *image_in_v,
               *image_out_r,
               *image_out_g,
               *image_out_b;
-main_variable_list_end

-main_before_lib_call
    if(check_args()) exit(1);

    READINPUT (image_in_y, image_in_u, image_in_v);
    CHECKINPUT (image_in_y, image_in_u, image_in_v);
-main_before_lib_call_end

```

```

-main_library_call
    if(!lyuv2rgb( image_in_y,  image_in_u,  image_in_v,
                  &image_out_r, &image_out_g, &image_out_b))
    {
        (void) fprintf(stderr,"lyuv2rgb failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if(! writeimage(yuv2rgb->or_file, image_out_r))
    {
        (void) fprintf(stderr,"yuv2rgb : unable to write r output file\n");
        exit(1);
    }
    if(! writeimage(yuv2rgb->og_file, image_out_g))
    {
        (void) fprintf(stderr,"yuv2rgb : unable to write g output file\n");
        exit(1);
    }
    if(! writeimage(yuv2rgb->ob_file, image_out_b))
    {
        (void) fprintf(stderr,"yuv2rgb : unable to write b output file\n");
        exit(1);
    }
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
    .IP "image_in_y" 15
    pointer to the luma   Y-Component.
    .IP "image_in_u" 15
    pointer to the chroma U-Component.
    .IP "image_in_v" 15
    pointer to the chroma V-Component.
    .IP "image_out_r" 15
    pointer to image pointer to receive the output image for R-Component.
    .IP "image_out_g" 15
    pointer to image pointer to receive the output image for G-Component.
    .IP "image_out_b" 15
    pointer to image pointer to receive the output image for B-Component.
-library_input_end

-library_output
    .IP "image_out_r" 15
    pointer to image pointer to receive the output image for R-Component.
    .IP "image_out_g" 15
    pointer to image pointer to receive the output image for G-Component.
    .IP "image_out_b" 15
    pointer to image pointer to receive the output image for B-Component.
    .IP "Return Value:" 15
    1 on success, 0 on failure.
-library_output_end

-library_def
int lyuv2rgb(image_in_y, image_in_u, image_in_v,
            image_out_r, image_out_g, image_out_b)

struct xvimage *image_in_y, *image_in_u, *image_in_v,
                **image_out_r, **image_out_g, **image_out_b;
-library_def_end

-library_code
/*****
/*
/* Library Code for yuv2rgb
/*

```

```

/*****/
{
float          f_l, f_u, f_v,
               f_r, f_g, f_b;
unsigned char  *l_in,  *u_in,  *v_in,
               *r_out, *g_out, *b_out;
unsigned long  x,                      /* col */
               y,                      /* row */
               rowsize,                /* of input images */
               colsize;                /* of input images */

/*****/

fprintf (stdout, "\nYUV2RGB V1.0:\n");

colsize = image_in_y->col_size;
rowsize = image_in_y->row_size;

/*****/
/* Allocate memory for the output images */
/*****/

*image_out_r = createimage(colsize, rowsize, VFF_TYP_1_BYTE, 1, 1,
                           "created by yuv2rgb", 0, 0, VFF_MS_NONE,
                           VFF_MAPTYP_NONE, VFF_LOC_IMPLICIT, 0);
*image_out_g = createimage(colsize, rowsize, VFF_TYP_1_BYTE, 1, 1,
                           "created by yuv2rgb", 0, 0, VFF_MS_NONE,
                           VFF_MAPTYP_NONE, VFF_LOC_IMPLICIT, 0);
*image_out_b = createimage(colsize, rowsize, VFF_TYP_1_BYTE, 1, 1,
                           "created by yuv2rgb", 0, 0, VFF_MS_NONE,
                           VFF_MAPTYP_NONE, VFF_LOC_IMPLICIT, 0);
if(*image_out_r == NULL || *image_out_g == NULL || *image_out_b == NULL)
{
    (void) fprintf(stderr, "lyuv2rgb : unable to allocate memory\n");
    return (0);
}

/*****/
/* Cast to the proper data type */
/*****/

l_in = (unsigned char *) image_in_y->imagedata;
u_in = (unsigned char *) image_in_u->imagedata;
v_in = (unsigned char *) image_in_v->imagedata;
r_out = (unsigned char *) (*image_out_r)->imagedata;
g_out = (unsigned char *) (*image_out_g)->imagedata;
b_out = (unsigned char *) (*image_out_b)->imagedata;

/*****/
/* Matrixing YUV to RGB */
/*****/

for (y = 0L; y < rowsize; y++)
{
    for (x = 0L; x < colsize; x++)
    {
        f_l = (float) *l_in; f_u = (float) *u_in; f_v = (float) *v_in;

        f_l = (f_l - 16.0) / 219.0;
        f_u = (f_u - 128.0) / 112.0 * 0.437;
        f_v = (f_v - 128.0) / 112.0 * 0.615;

        f_r =          f_l          + 1.140 * f_v;
        f_g =          f_l - 0.394 * f_u - 0.581 * f_v;
        f_b = 0.998 * f_l + 2.028 * f_u;

        if (f_r > 1.0) f_r = 1.0;      if (f_r < 0.0) f_r = 0.0;
        if (f_g > 1.0) f_g = 1.0;      if (f_g < 0.0) f_g = 0.0;
        if (f_b > 1.0) f_b = 1.0;      if (f_b < 0.0) f_b = 0.0;
    }
}

```

```

        *r_out = (unsigned char) (f_r * 255.0) ;
        *g_out = (unsigned char) (f_g * 255.0);
        *b_out = (unsigned char) (f_b * 255.0);

        l_in++; u_in++; v_in++;
        r_out++; g_out++; b_out++;
    }

    fprintf (stdout,"Done: YUV2RGB\n");

return (1);
}
-library_code_end

-library_mods
-library_mods_end

```

A.5 peaking

A.5.1 peaking.pane

```

-F 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +0+0 'Enhance the overall sharpness of Component Signals' peaking
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input Signal' 'Input Signal' i
-O 1 0 0 1 0 0 50x1+2+4 +0+0 ' ' 'Output Signal' 'Output Signal' o
-b +2+6 'Parameters for weighting of signals (0...100):'
-f 1 0 0 1 0 12x1+25+7 +1+0 0 100 1 'ah' 'horizontal multiplier' ah
-f 1 0 0 1 0 12x1+2+7 +1+0 0 100 1 'av' 'vertical multiplier' av
-f 1 0 0 1 0 12x1+25+8 +1+0 0 100 1 'ch' 'horizontal coring' ch
-f 1 0 0 1 0 12x1+2+8 +1+0 0 100 1 'cv' 'vertical coring' cv
-T 1 0 0 1 0 40x1+2+10 +0+0 3 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 0 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-R 1 0 1 10x2+1+13 'Execute' 'do operation' $PALPLUS/bin/peaking
-Q 1 0 10x2+20+13 'Quit'
-H 1 10x2+39+13 'Help' 'man page for peaking' $PALPLUS/doc/manpages/peaking.1
-E
-E
-E

```

A.5.2 peaking.prog

```

-authors
Harald Zottmann
-authors_end

-short_prog_description
Enhance the sharpness in a component signal.
-short_prog_description_end

-short_lib_description
Enhance the sharpness in a component signal.
-short_lib_description_end

-man1_long_description
.I peaking
will enhance the sharpness in a component signal.
.LP
The peaking filter with coring will amplifie the vertical and
horizontal high frequencies to improve the subjective perception

```

of sharpness.

.LP

The input signal will be filtered with a vertical high-pass filter, to get vertical detailinformation, and with a horizontal high-pass filter, to get horizontal detailinformation. Both resulting high frequency correction signals will pass non-linear weightings and then will be added to the original signal.

.LP

The parameters for the non-linear weightings are ah,av and ch,cv. The suffix h and v are necessary to distinguish between horizontal and vertical signal path.

.LP

The parameter c (ch,cv) sets the coring of the detailinformation signals. This means, that all values below c are set to zero.

.LP

```

Non-Linear Weighting:  |  /
                      |  /
                    -c |  /
<----->
                /  |  c
                /  |
                /  |

```

.LP

The parameter a (ah,av) is used as a multiplier for the detailinformation signals, before they are added to the original signal.

.LP

(Schoenfelder,H.: Digitale Filter in der Videotechnik. 1.Aufl.-Berlin: Drei-R-Verlag (1988), pp 246-248)

-man1_long_description_end

-man1_examples

-man1_examples_end

-man1_restrictions

.I peaking

.LP

Datatypes of input and output signals:

.LP

```

input signal:      1_BYTE (0...255)
output signal:     1_BYTE (0...255)

```

-man1_restrictions_end

-man1_see_also

-man1_see_also_end

-man3_long_description

.I peaking

will enhance the sharpness in a component signal.

.LP

The peaking filter with coring will amplifie the vertical and horizontal high frequencies to improve the subjective perception of sharpness.

.LP

The input signal will be filtered with a vertical high-pass filter, to get vertical detailinformation, and with a horizontal high-pass filter, to get horizontal detailinformation. Both resulting high frequency correction signals will pass non-linear weightings and then will be added to the original signal.

.LP

The parameters for the non-linear weightings are ah,av and ch,cv. The suffix h and v are necessary to distinguish between horizontal and vertical signal path.

.LP

The parameter c (ch,cv) sets the coring of the detailinformation signals. This means, that all values below c are set to zero.

.LP

The parameter a (ah,av) is used as a multiplier for the detailinformation signals, before they are added to the original signal.

-man3_long_description_end

```

-man3_restrictions
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

-usage_additions
-usage_additions_end

-include_includes
-include_includes_end

-include_additions
-include_additions_end

-include_macros
/*****
/* Read Input-Images */
*****/

#define READINPUT(in_img); \
    in_img = readimage(peaking->i_file);\
    if(in_img == NULL)\
    {\
        (void) fprintf(stderr,"peaking : cannot read input image");\
        (void) fprintf(stderr," %s\n",peaking->i_file);\
        exit(1);\
    }

/*****
/* Check Input-Images */
*****/

#define CHECKINPUT(in_img); \
    (void) proptype(program, in_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, in_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, in_img, 1, TRUE);\
    (void) proper_map_enable(program, in_img, VFF_MAP_OPTIONAL, TRUE);
-include_macros_end

-main_variable_list
struct xvimage  *in_img,
                *out_img,
                *readimage();
-main_variable_list_end

-main_before_lib_call
    if (check_args()) exit(1);

    READINPUT (in_img);
    CHECKINPUT (in_img);
-main_before_lib_call_end

-main_library_call
    if(! lpeaking(in_img,
                  &out_img,
                  peaking->av_float, peaking->ah_float,
                  peaking->cv_float, peaking->ch_float,
                  peaking->f_sc_toggle) )
    {
        (void) fprintf(stderr, "lpeaking failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if(! writeimage(peaking->o_file, out_img))
    {
        (void) fprintf(stderr,"peaking : unable to write output file\n");
        exit(1);
    }

```

```

    }
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lpeaking( in_img,
              out_img,
              av, ah,
              cv, ch,
              f_sc_toggle)

struct ximage *in_img,
              **out_img;

float        av, ah,
              cv, ch;

int          f_sc_toggle;
-library_def_end

-library_code
/*****
/*
/* Improve Sharpness
/*
/*****
/*****
/*
/* Library Code for peaking
/*
/*
/*****

#define FCARRIER      4433618.75
#define FROW           15625

#define FC_FAC_4       1
#define FC_FAC_8       2
#define FC_FAC_12      3

/*****
/*
/* Main Function
/*
/*
/*****

{
float        signal          = 0.0,
              carrier_fac    = 0.0;
unsigned char *in_ptr        = NULL,
              *out_ptr       = NULL;
signed char  *c_b_temp      = NULL,
              *c_r_temp      = NULL;
signed long  il,
              in_length      = 0L,
              pixel_1row     = 0L,
              pixel_2row     = 0L;

/*****

fprintf(stdout, "\nPEAKING V1.1:\n");

switch (f_sc_toggle)

```

```

    {
    case FC_FAC_4:
        carrier_fac = 4.0;
        break;

    case FC_FAC_8:
        carrier_fac = 8.0;
        break;

    case FC_FAC_12:
        carrier_fac = 12.0;
        break;

    default:
        fprintf (stderr, "Missing or improper argument in call to peaking.\n");
        return (0);
    }

    pixel_1row = (signed long) (1.0 * FCARRIER / FROW * carrier_fac);
    pixel_2row = (signed long) (2.0 * FCARRIER / FROW * carrier_fac);

    /*****

    in_ptr      = (unsigned char *) in_img->imagedata;
    in_length = (signed long) (in_img->row_size * in_img->col_size);

    /*****
    /* Allocate memory for Output-Image                                     */
    /*****

    *out_img = createimage ((unsigned long) (in_length), 1L, VFF_TYP_1_BYTE,
                           1,1, "created by peaking", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
                           VFF_LOC_IMPLICIT, 0);
    out_ptr = (unsigned char *) (*out_img)->imagedata;

    if (*out_img == NULL)
    {
        fprintf (stderr, "Out of memory.\n");
        return (0);
    }

    /*****
    /* Allocate and set temporary memory for signal-processing               */
    /*****

    c_b_temp = (signed char *)
                malloc((unsigned) (in_length + pixel_2row) * sizeof(char));

    c_r_temp = (signed char *)
                malloc((unsigned) (in_length + pixel_2row) * sizeof(char));

    if ((c_b_temp == NULL) || (c_r_temp == NULL))
    {
        if (c_b_temp != NULL) free(c_b_temp);
        if (c_r_temp != NULL) free(c_r_temp);
        fprintf (stderr, "Out of memory.\n");
        return (0);
    }

    for (il = 0L; il < in_length + pixel_2row; il++)
    {
        *(c_b_temp + il) = 0;
        *(c_r_temp + il) = 0;
    }

    /*****
    /* Vertical                                                             */
    /*****

    for (il = 0L; il < in_length; il++)

```

```

    {
        *(c_r_temp + il)
            -= (signed char) (0.25 * (float) *(in_ptr + il));
        *(c_r_temp + il + pixel_1row)
            += (signed char) (0.50 * (float) *(in_ptr + il));
        *(c_r_temp + il + pixel_2row)
            -= (signed char) (0.25 * (float) *(in_ptr + il));
    }

/*****
/* Horizontal */
*****/

    for (il = 0L; il < in_length; il++)
    {
        *(c_b_temp + il + pixel_1row - 1L)
            -= (signed char) (0.25 * (float) *(in_ptr + il));
        *(c_b_temp + il + pixel_1row)
            += (signed char) (0.50 * (float) *(in_ptr + il));
        *(c_b_temp + il + pixel_1row + 1L)
            -= (signed char) (0.25 * (float) *(in_ptr + il));
    }

/*****
/* Shift back Signals */
*****/

    for (il = 0L; il < in_length; il++)
    {
        *(c_r_temp + il) = *(c_r_temp + il + pixel_1row);
        *(c_b_temp + il) = *(c_b_temp + il + pixel_1row);
    }

/*****
/* Non-Linear-Weighting */
*****/

    for (il = 0L; il < in_length; il++)
    {
        signal = (float) *(c_r_temp + il);
        if (signal > 0.0) /* positive */
        {
            if (signal <= cv) signal = 0.0;
            else signal = signal - cv;
        }
        else /* negative */
        {
            if (signal >= (-1.0 * cv)) signal = 0.0;
            else signal = signal + cv;
        }
        if (signal > 127.0) signal = 127.0;
        if (signal < -127.0) signal = -127.0;

        *(c_r_temp + il) = (signed char) signal;
    }

/*****

    for (il = 0L; il < in_length; il++)
    {
        signal = (float) *(c_b_temp + il);
        if (signal > 0.0) /* positive */
        {
            if (signal <= ch) signal = 0.0;
            else signal = signal - ch;
        }
        else /* negative */
        {
            if (signal >= (-1.0 * ch)) signal = 0.0;
            else signal = signal + ch;
        }
    }

```

```

    }
    if (signal > 127.0)    signal = 127.0;
    if (signal < -127.0)  signal = -127.0;

    *(c_b_temp + il) = (signed char) signal;
}

/*****
/* Add all Signals to get Ouput-Signal
*****/

for (il = 0L; il < in_length; il++)
{
    signal = ((float) *(in_ptr + il) )
            + ((float) *(c_b_temp + il) * ah)
            + ((float) *(c_r_temp + il) * av);

    if (signal > 255.0)    signal = 255.0;
    if (signal < 0.0)     signal = 0.0;

    *(out_ptr + il) = (unsigned char) signal;
}

fprintf(stdout, "Done: PEAKING\n");

return (1);
}
-library_code_end

-library_mods
-library_mods_end

```

A.6 cti

A.6.1 cti.pane

```

-F 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +1+0 'Color Transient Improvement (CTI)' cti
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input Signal' 'Input Signal' i
-O 1 0 0 1 0 0 50x1+2+4 +0+0 ' ' 'Output Signal' 'Output Signal' o
-T 1 0 0 1 0 40x1+2+6 +0+0 1 'Power of polynomials for use in edge enhancement:' 'Power' power
-l 1 0 1 0 0 1x1+2+4 +0+0 0 ' ' 'False' 'True' 'Third order polynomial' logical
-l 1 0 1 1 0 1x1+2+1 +0+0 0 'Power = 2' 'False' 'True' 'Power = 2' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 'Power = 3' 'False' 'True' 'Power = 3' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 'Power = 4' 'False' 'True' 'Power = 4' logical
-l 1 0 1 0 0 1x1+41+1 +0+0 0 'Power = 5' 'False' 'True' 'Power = 5' logical
-E
-b +2+9 'Third order polynomials (Position of the polynomial minimum):'
-f 1 0 0 1 0 45x1+7+10 +0+0 0 0.288675 0.2 'Position' 'Position of polynomial minimum' pmin
-b +2+12 'Bandwith of signal to enhance (MHz):'
-f 1 0 0 1 0 50x1+2+13 +0+0 0.1 5 1 'Minimum' 'Minimum' min
-f 1 0 0 1 0 50x1+2+14 +0+0 0.1 5 1.8 'Maximum' 'Maximum' max
-T 1 0 0 1 0 40x1+2+16 +0+0 1 'Select Output Signal:' 'Output Signal' output
-l 1 0 1 1 0 1x1+2+1 +0+0 0 'Improved Signal' 'False' 'True' 'Improved Signal' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 'Control Signal' 'False' 'True' 'Control Signal' logical
-E
-T 1 0 0 1 0 40x1+2+19 +0+0 3 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 0 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-R 1 0 1 10x2+1+22 'Execute' 'do operation' $PALPLUS/bin/cti
-Q 1 0 10x2+20+22 'Quit'
-H 1 10x2+39+22 'Help' 'man page for cti' $PALPLUS/doc/manpages/cti.1
-E

```

-E
-E

A.6.2 cti.prog

```
-authors
Harald Zottmann
-authors_end

-short_prog_description
Edge enhancement algorithm using a polynomial transformation
(Color Transient Improvement).
-short_prog_description_end

-short_lib_description
Edge enhancement algorithm using a polynomial transformation
(Color Transient Improvement).
-short_lib_description_end

-man1_long_description
.I cti
will improve the transients in a color component signal using
a polynomial transformation.
.LP
The input signal has to be a color component signal (u/v-component).
Normally the improved signal should be selected for the output signal.
In order to have the possibility to check which areas of the signal will
be improved, the control signal for the enhancement algorithm can be
directed to the output signal. The resulting image of the control signal
knows three values (1_BYTE):
.LP
    0 (black): Negative transient to be improved.
    127 (grey): Slow varying intensity area.
    255 (white): Positive transient to be improved.
.LP
The minimum bandwidth defines the longest transient to be improved.
The lower the bandwidth, the longer the longest possible transient (B*T=1).
The maximum bandwidth defines the shortest transient to be improved.
The higher the bandwidth, the shorter the shortest possible transient.
This value may not be too high, in order not to improve noisy peaks as
well as the signal.
.LP
The power of the polynomials for use in the edge enhancement algorithm
can be chosen from 2 to 5. The higher the power, the sharper the improved
transient. Another kind of polynomial which can be used is the third order
polynomial. If selected, the parameter 'position' marks the polynomial
minimum. The higher the value of 'position' the sharper the enhanced edge.
.LP
(Fonsalas,F.; Lejard,J.Y.: An Edge Enhancement Algorithm Using a
Polynomial Transformation. :IEEE Transactions on Consumer Electronics
Vol.CE-32 (1986), No.3, pp 151-154)
-man1_long_description_end

-man1_examples
-man1_examples_end

-man1_restrictions
.I cti
.LP
Datatypes of input and output signals:
.LP
    input signal:          1_BYTE (16...240)
    output signal:         1_BYTE (16...240)
-man1_restrictions_end

-man1_see_also
-man1_see_also_end
```

```

-man3_long_description
.I cti
will improve the transients in a color component signal using
a polynomial transformation.
.LP
The input signal has to be a color component signal (u/v-component).
Normally the improved signal should be selected for the output signal.
In order to have the possibility to check which areas of the signal will
be improved, the control signal for the enhancement algorithm can be
directed to the output signal. The resulting image of the control signal
knows three values (1_BYTE):
.LP
    0 (black): Negative transient to be improved.
    127 (grey): Slow varying intensity area.
    255 (white): Positive transient to be improved.
.LP
The minimum bandwidth defines the longest transient to be improved.
The lower the bandwidth, the longer the longest possible transient (B*T=1).
The maximum bandwidth defines the shortest transient to be improved.
The higher the bandwidth, the shorter the shortest possible transient.
This value may not be too high, in order not to improve noisy peaks as
well as the signal.
.LP
The power of the polynomials for use in the edge enhancement algorithm
can be chosen from 2 to 5. The higher the power, the sharper the improved
transient. Another kind of polynomial which can be used is the third order
polynomial. If selected, the parameter 'position' marks the polynomial
minimum. The higher the value of 'position' the sharper the enhanced edge.
-man3_long_description_end

-man3_restrictions
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

-usage_additions
-usage_additions_end

-include_includes
-include_includes_end

-include_additions
-include_additions_end

-include_macros
/*****
/* Read Input-Images */
*****/

#define READINPUT(in_img); \
    in_img = readimage(cti->i_file);\
    if(in_img == NULL)\
    {\
        (void) fprintf(stderr,"cti : cannot read input image");\
        (void) fprintf(stderr," %s\n",cti->i_file);\
        exit(1);\
    }

/*****
/* Check Input-Images */
*****/

#define CHECKINPUT(in_img); \
    (void) proptype(program, in_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, in_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, in_img, 1, TRUE);\
    (void) proper_map_enable(program, in_img, VFF_MAP_OPTIONAL, TRUE);
-include_macros_end

```

```

-main_variable_list
struct xvimage *in_img,
               *out_img,
               *readimage();
-main_variable_list_end

-main_before_lib_call
    if (check_args()) exit(1);

    READINPUT (in_img);
    CHECKINPUT (in_img);
-main_before_lib_call_end

-main_library_call
    if(! lcti(in_img,
              &out_img,
              cti->min_float,
              cti->max_float,
              cti->pmin_float,
              cti->power_toggle,
              cti->output_toggle,
              cti->f_sc_toggle) )
    {
        (void) fprintf(stderr, "lcti failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if(! writeimage(cti->o_file, out_img))
    {
        (void) fprintf(stderr, "cti : unable to write output file\n");
        exit(1);
    }
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lcti( in_img,
         out_img,
         min, max,
         pmin,
         power_toggle,
         output_toggle,
         f_sc_toggle)

struct xvimage *in_img,
               **out_img;

float          min, max,
               pmin;

int            power_toggle,
               output_toggle,
               f_sc_toggle;
-library_def_end

-library_code
/*****
/*
/* Improve Sharpness
/*

```

```

/*****
/*****
/*
/* Library Code for Color Transient Improvement (CTI)
/*
/*
/*****

#define FCARRIER      4433618.75
#define FROW           15625

#define FC_FAC_4       1
#define FC_FAC_8       2
#define FC_FAC_12      3

#define THIRDORDER     1
#define POWER_2        2
#define POWER_3        3
#define POWER_4        4
#define POWER_5        5

#define IMPR_SIG        1
#define CTRL_SIG        2

/*****
/*
/* Main Function
/*
/*
/*****

{
void      polynomial      ();

float      signal          = 0.0,
            carrier_fac    = 0.0,
            samp_freq      = 0.0,
            min_bw         = 0.0,      /* Min. Bandwidth */
            max_bw         = 0.0,      /* Max. Bandwidth */
            power          = 0.0;

unsigned char *in_ptr      = NULL,
              *out_ptr     = NULL,
              *pol_ptr     = NULL,      /* Where to insert a polynomial */
              amp          = 0,        /* Startvalue of polynomial */
              offset       = 0;        /* Endvalue of polynomial */

signed char *deriv_temp   = NULL,
            old_deriv     = 0;        /* Old value of derivation */

signed long il, kl,
            count         = 0L,      /* Number of values of same sign */
            in_length     = 0L,
            min_length    = 0L,      /* Min. Length for improvement */
            max_length    = 0L;      /* Max. Length for improvement */

/*****

fprintf(stdout, "\nCTI V1.2:\n");

switch (f_sc_toggle)
{
    case FC_FAC_4:
        carrier_fac = 4.0;
        break;

    case FC_FAC_8:
        carrier_fac = 8.0;
        break;

    case FC_FAC_12:
        carrier_fac = 12.0;
        break;

    default:

```

```

        fprintf (stderr,"Missing or improper argument in call to cti.\n");
        return (0);
    }
    samp_freq = FCARRIER * carrier_fac;

    switch (power_toggle)
    {
        case THIRDDORDER:
            power = 0.0;
            break;

        case POWER_2:
            power = 2.0;
            break;

        case POWER_3:
            power = 3.0;
            break;

        case POWER_4:
            power = 4.0;
            break;

        case POWER_5:
            power = 5.0;
            break;

        default:
            fprintf (stderr,"Missing or improper argument in call to cti.\n");
            return (0);
    }

    if (max <= min)
    {
        fprintf (stderr,"Maximum bandwidth must be greater than Minimum bandwidth.\n");
        return (0);
    }
    min_bw = 1.0E6 * min;
    max_bw = 1.0E6 * max;

    max_length = (signed long) (samp_freq / 2.0 / min_bw);
    if (((samp_freq / 2.0 / min_bw) - (float) max_length) >= 0.5)
    {
        max_length ++;
    }
    min_length = (signed long) (samp_freq / 2.0 / max_bw);
    if (((samp_freq / 2.0 / max_bw) - (float) min_length) >= 0.5)
    {
        min_length ++;
    }

    fprintf (stdout,"Transient range goes from %ld to %ld pixels\n",
            min_length, max_length);

    /*****

    in_ptr    = (unsigned char *) in_img->imagedata;
    in_length = (signed long) (in_img->row_size * in_img->col_size);

    /*****
    /* Allocate memory for Output-Image */
    /*****

    *out_img = createimage ((unsigned long) (in_length), 1L, VFF_TYP_1_BYTE,
        1,1, "created by cti", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
        VFF_LOC_IMPLICIT, 0);
    out_ptr = (unsigned char *) (*out_img)->imagedata;

    if (*out_img == NULL)
    {

```

```

        fprintf (stderr, "Out of memory.\n");
        return (0);
    }

/*****
/* Allocate and set temporary memory for signal-processing */
*****/

    deriv_temp = (signed char *)
        malloc((unsigned) in_length * sizeof(char));

    if (deriv_temp == NULL)
    {
        fprintf (stderr, "Out of memory.\n");
        return (0);
    }

/*****
/* Get the signal for the sign of the derivative */
*****/

    for (il = 1L; il < in_length; il++)
    {
        signal = (float) *(in_ptr + il) - (float) *(in_ptr + il - 1);
        if (signal < 0.0) *(deriv_temp + il) = (signed char) -1;
        else if (signal > 0.0) *(deriv_temp + il) = (signed char) +1;
        else *(deriv_temp + il) = (signed char) 0;
    }

/*****
/* Get the control signal from the signal for the sign of the derivative */
*****/

    for (il = 0L; il < in_length; il++)
    {
        if (old_deriv == *(deriv_temp + il))
        {
            count++;
        }
        else
        {
            if ((count > max_length) || (count < min_length))
            {
                for (kl = il - 1L; kl >= il - count; kl--)
                {
                    *(deriv_temp + kl) = (signed char) 0;
                }
            }
            old_deriv = *(deriv_temp + il);
            count = 1L;
        }
    }

/*****

    if (output_toggle == CTRL_SIG)
    {
        fprintf (stdout, "Control Signal\n");

        for (il = 0L; il < in_length; il++)
        {
            signal = *(deriv_temp + il);

            if (signal == 0) signal = 127.0; /* No polynomial */
            else if (signal == 1) signal = 255.0; /* Rising polynomial */
            else signal = 0.0; /* Falling polynomial */

            *(out_ptr + il) = signal;
        }
    }

```

```

/*****/

else
{
    fprintf (stdout, "Improved Signal\n");

    count      = 0L;
    old_deriv   = 0;

    for (il = 0L; il < in_length; il++)
    {
        *(out_ptr + il) = *(in_ptr + il);      /* Copy Input to Output */

        if (old_deriv == *(deriv_temp + il)) /* CTI */
        {
            count++;
        }
        else
        {
            if (old_deriv != 0)
            {
                if (old_deriv == 1)
                {
                    offset = *(in_ptr + il - count - 1L);
                    amp     = *(in_ptr + il - 1L);
                    amp     -= offset;
                }
                else if (old_deriv == -1)
                {
                    offset = *(in_ptr + il - 1L);
                    amp     = *(in_ptr + il - count - 1L);
                    amp     -= offset;
                }
                pol_ptr = (unsigned char *) (out_ptr + il - count - 1L);

                polynomial (pol_ptr, count, power, pmin,
                           old_deriv, offset, amp);
            }
            old_deriv = *(deriv_temp + il);
            count = 1L;
        }
    }
}

/*****/

free (deriv_temp);

fprintf(stdout, "Done: CTI\n");

return (1);
}

/*****/

void polynomial (ptr, length, power, pmin, deriv, offset, amp)
unsigned char *ptr;
signed long length;
float power,
pmin;
signed char deriv;
unsigned char offset,
amp;
{
    float a = 0.0,
          b = 0.0,
          v = 0.0,
          x = 0.0,
          c = 0.0;

```

```

signed long    il;

    if (length <= 0)
    {
        return;
    }

/*****

    if (power == 0.0)        /* Third order polynomial */
    {
        a = 4.0 / (1.0 - 12.0 * pmin * pmin);
        b = -3.0 * pmin * pmin;

        for (il = 0; il <= length; il++)
        {
            x = (float) il / (float) length;

            if (x < 0.5)
            {
                v = (float) (      (a * (pow (x, 3.0) + b * x)));
            }

            else
            {
                x = 1.0 - x;
                v = (float) (1.0 - (a * (pow (x, 3.0) + b * x)));
            }

            if (deriv == -1)
            {
                v = 1.0 - v;
            }

            v *= (float) amp;
            v += (float) offset;

            if (v < 0.0) v = 0.0;
            if (v > 255.0) v = 255.0;

            *(ptr + il) = (unsigned char) v;
        }
    }

/*****

    else                                /* Power functions */
    {
        c = pow (2.0, power - 1.0);

        for (il = 0; il <= length; il++)
        {
            x = (float) il / (float) length;

            if (x < 0.5)
            {
                v = (float) (      c * pow (x, power));
            }

            else
            {
                x = 1.0 - x;
                v = (float) (1.0 - c * pow (x, power));
            }

            if (deriv == -1)
            {
                v = 1.0 - v;
            }

            v *= (float) amp;
            v += (float) offset;

            if (v < 0.0) v = 0.0;
            if (v > 255.0) v = 255.0;

```

```

        *(ptr + il) = (unsigned char) v;
    }

    return;
}
-library_code_end

-library_mods
-library_mods_end

```

A.7 medfil

A.7.1 medfil.pane

```

-F 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +0+0 'Median Filter for demodulated Component Signals' medfil
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input Component Signal' 'Input Component Signal' i
-O 1 0 0 1 0 0 50x1+2+4 +0+0 ' ' 'Output Component Signal' 'Output Component Signal' o
-b +2+6 'Dimension of Median Filter:'
-l 1 0 0 1 0 50x1+2+7 +0+0 1 'Horizontal' 'OFF' 'ON' 'Horizontal' hor
-l 1 0 0 1 0 50x1+2+8 +0+0 1 'Vertical' 'OFF' 'ON' 'Vertical' ver
-l 1 0 0 1 0 50x1+2+9 +0+0 1 'Time' 'OFF' 'ON' 'Time' tim
-T 1 0 0 1 0 40x1+2+11 +0+0 1 'Order of Median Filter:' 'order' order
-l 1 0 1 1 0 1x1+2+1 +0+0 0 'Order = 1' 'False' 'True' 'Order = 1' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 'Order = 2' 'False' 'True' 'Order = 2' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 'Order = 3' 'False' 'True' 'Order = 3' logical
-l 1 0 1 0 0 1x1+41+1 +0+0 0 'Order = 4' 'False' 'True' 'Order = 4' logical
-E
-T 1 0 0 1 0 40x1+2+14 +0+0 3 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 0 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-R 1 0 1 10x2+1+17 'Execute' 'do operation' $HOME/medfil/medfil
-Q 1 0 10x2+20+17 'Quit'
-H 1 10x2+39+17 'Help' 'man page for medfil' $PALPLUS/doc/manpages/medfil.1
-E
-E
-E

```

A.7.2 medfil.prog

```

-authors
Harald Zottmann
-authors_end

-short_prog_description
Median-Filter for demodulated component signals.
-short_prog_description_end

-short_lib_description
Median-Filter for demodulated component signals.
-short_lib_description_end

-man1_long_description
.I medfil
will free demodulated component signals from noise, using
a median-filter.
.LP
The median-filter is a non-linear filter with low-pass quality.
It selects the mean value out of a couple of samples which are
located next to each other (x,y,t). Therefore the output value
always is a value from the original input signal.

```

The median-filter can be used to free a signal from noise. It is very effective on impuls-like noise.

.LP

The order of the median-filter defines, from how many samples the mean value will be selected. The dimension selects, in which direction the filter will work.

.LP

(Wischermann,G.: Medianfilterung von Videosignale.

FKT, Vol. 44 (1990), No.12, pp 665-672)

-man1_long_description_end

-man1_examples

-man1_examples_end

-man1_restrictions

.I medfil

.LP

Datatypes of input and output signals:

.LP

input signal: 1_BYTE (0...255)

output signal: 1_BYTE (0...255)

-man1_restrictions_end

-man1_see_also

-man1_see_also_end

-man3_long_description

.I medfil

will free demodulated component signals from noise, using a median-filter.

.LP

The median-filter is a non-linear filter with low-pass quality. It selects the mean value out of a couple of samples which are located next to each other (x,y,t). Therefore the output value always is a value from the original input signal.

The median-filter can be used to free a signal from noise. It is very effective on impuls-like noise.

.LP

The order of the median-filter defines, from how many samples the mean value will be selected. The dimension selects, in which direction the filter will work.

-man3_long_description_end

-man3_restrictions

-man3_restrictions_end

-man3_see_also

-man3_see_also_end

-usage_additions

-usage_additions_end

-include_includes

-include_includes_end

-include_additions

-include_additions_end

-include_macros

```

/*****
/* Read Input-Image */
*****/

```

```

#define READINPUT(in_img); \
    in_img = readimage(medfil->i_file);\
    if(in_img == NULL)\
    {\
        (void) fprintf(stderr,"medfil : cannot read input image");\
        (void) fprintf(stderr," %s\n",medfil->i_file);\
        exit(1);\
    }

```

```

    }

/*****
/* Check Input-Image
*****/

#define CHECKINPUT(in_img); \
    (void) proptype(program, in_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, in_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, in_img, 1, TRUE);\
    (void) proper_map_enable(program, in_img, VFF_MAP_OPTIONAL, TRUE);
#include_macros_end

-main_variable_list
struct xvimage *in_img,
               *out_img,
               *readimage();
-main_variable_list_end

-main_before_lib_call
    if (check_args()) exit(1);

    READINPUT (in_img);
    CHECKINPUT (in_img);
-main_before_lib_call_end

-main_library_call
    if(! lmedfil(in_img,
                &out_img,
                medfil->hor_logic,
                medfil->ver_logic,
                medfil->tim_logic,
                medfil->f_sc_toggle,
                medfil->order_toggle) )
    {
        (void) fprintf(stderr, "lmedfil failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if(! writeimage(medfil->o_file, out_img))
    {
        (void) fprintf(stderr, "medfil : unable to write output file\n");
        exit(1);
    }
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lmedfil( in_img,
            out_img,
            hor, ver, tim,
            f_sc_toggle,
            order)

struct xvimage *in_img,
               **out_img;

int
    hor, ver, tim,
    f_sc_toggle,
    order;

```

```

-library_def_end

-library_code
/*****
/*
/* Median-Filter
/*
/*
/*****
/*****
/*
/* Library Code for medfil
/*
/*
/*****

#define FCARRIER      4433618.75
#define FROW           15625

#define FC_FAC_4       1
#define FC_FAC_8       2
#define FC_FAC_12      3

#define ORDER_1        1
#define ORDER_2        2
#define ORDER_3        3
#define ORDER_4        4

/*****
/*
/* Main Function
/*
/*
/*****

{
unsigned char   get_median      ();

float           carrier_fac     = 0.0;
unsigned char   *in_ptr         = NULL,
                *out_ptr        = NULL,
                med_elem [25];
signed long     il,
                in_length      = 0L,
                pixel_1row     = 0L,
                pixel_2row     = 0L,
                pixel_3row     = 0L,
                pixel_4row     = 0L,
                pixel_1frame    = 0L,
                pixel_2frame    = 0L,
                pixel_3frame    = 0L,
                pixel_4frame    = 0L;
int             med_pos        = 0;

/*****

fprintf(stdout, "\nMEDFIL V1.0:\n");

switch (f_sc_toggle)
{
    case FC_FAC_4:
        carrier_fac = 4.0;
        break;

    case FC_FAC_8:
        carrier_fac = 8.0;
        break;

    case FC_FAC_12:
        carrier_fac = 12.0;
        break;

    default:

```

```

    fprintf(stderr, "Missing or improper argument in call to medfil.\n");
    return (0);
}

pixel_1row = (long) (1.0 * FCARRIER / FROW * carrier_fac);      /* 1T_H */
pixel_2row = (long) (2.0 * FCARRIER / FROW * carrier_fac);      /* 2T_H */
pixel_3row = (long) (3.0 * FCARRIER / FROW * carrier_fac);      /* 3T_H */
pixel_4row = (long) (4.0 * FCARRIER / FROW * carrier_fac);      /* 4T_H */

pixel_1frame = (long) (1.0 * FCARRIER / 25.0 * carrier_fac);     /* 1T_B */
pixel_2frame = (long) (2.0 * FCARRIER / 25.0 * carrier_fac);     /* 2T_B */
pixel_3frame = (long) (3.0 * FCARRIER / 25.0 * carrier_fac);     /* 3T_B */
pixel_4frame = (long) (4.0 * FCARRIER / 25.0 * carrier_fac);     /* 4T_B */

/*****

in_ptr    = (unsigned char *) in_img->imagedata;
in_length = (signed long) (in_img->row_size * in_img->col_size);

/*****
/* Allocate memory for Output-Image */
/*****

*out_img = createimage ((unsigned long) (in_length), 1L, VFF_TYP_1_BYTE,
                        1,1, "created by medfil", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
                        VFF_LOC_IMPLICIT, 0);
out_ptr = (unsigned char *) (*out_img)->imagedata;

if (*out_img == NULL)
{
    fprintf(stderr, "Out of memory.\n");
    return (0);
}

/*****

if (!hor && !ver && !tim)
{
    for (il = 0L; il < in_length; il++)
    {
        *(out_ptr + il) = *(in_ptr + il);
    }
    fprintf(stdout, "Done: MEDFIL\n");
    return (1);
}

/*****

for (il = 0L; il < in_length; il++)
{
    med_elem[0] = *(in_ptr + il);
    med_pos = 1;

    if (hor)                                /* Horizontal */
    {
        if ((il >= 1L) && (il < in_length - 1L))
        {
            med_elem[med_pos] = *(in_ptr + il - 1L);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + 1L);
            med_pos++;
        }
        if (order >= ORDER_2)
        {
            if ((il >= 2L) && (il < in_length - 2L))
            {
                med_elem[med_pos] = *(in_ptr + il - 2L);
                med_pos++;
                med_elem[med_pos] = *(in_ptr + il + 2L);
                med_pos++;
            }

```

```

    }
    }
    if (order >= ORDER_3)
    {
        if ((il >= 3L) && (il < in_length - 3L))
        {
            med_elem[med_pos] = *(in_ptr + il - 3L);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + 3L);
            med_pos++;
        }
    }
    if (order >= ORDER_4)
    {
        if ((il >= 4L) && (il < in_length - 4L))
        {
            med_elem[med_pos] = *(in_ptr + il - 4L);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + 4L);
            med_pos++;
        }
    }
}

if (ver) /* Vertical (Row) */
{
    if ((il >= pixel_1row) && (il < in_length - pixel_1row))
    {
        med_elem[med_pos] = *(in_ptr + il - pixel_1row);
        med_pos++;
        med_elem[med_pos] = *(in_ptr + il + pixel_1row);
        med_pos++;
    }
    if (order >= ORDER_2)
    {
        if ((il >= pixel_2row) && (il < in_length - pixel_2row))
        {
            med_elem[med_pos] = *(in_ptr + il - pixel_2row);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + pixel_2row);
            med_pos++;
        }
    }
    if (order >= ORDER_3)
    {
        if ((il >= pixel_3row) && (il < in_length - pixel_3row))
        {
            med_elem[med_pos] = *(in_ptr + il - pixel_3row);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + pixel_3row);
            med_pos++;
        }
    }
    if (order >= ORDER_4)
    {
        if ((il >= pixel_4row) && (il < in_length - pixel_4row))
        {
            med_elem[med_pos] = *(in_ptr + il - pixel_4row);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + pixel_4row);
            med_pos++;
        }
    }
}

if (tim) /* Time (Frame) */
{
    if ((il >= pixel_1frame) && (il < in_length - pixel_1frame))
    {
        med_elem[med_pos] = *(in_ptr + il - pixel_1frame);
    }
}

```

```

        med_pos++;
        med_elem[med_pos] = *(in_ptr + il + pixel_1frame);
        med_pos++;
    }
    if (order >= ORDER_2)
    {
        if ((il >= pixel_2frame) && (il < in_length - pixel_2frame))
        {
            med_elem[med_pos] = *(in_ptr + il - pixel_2frame);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + pixel_2frame);
            med_pos++;
        }
    }
    if (order >= ORDER_3)
    {
        if ((il >= pixel_3frame) && (il < in_length - pixel_3frame))
        {
            med_elem[med_pos] = *(in_ptr + il - pixel_3frame);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + pixel_3frame);
            med_pos++;
        }
    }
    if (order >= ORDER_4)
    {
        if ((il >= pixel_4frame) && (il < in_length - pixel_4frame))
        {
            med_elem[med_pos] = *(in_ptr + il - pixel_4frame);
            med_pos++;
            med_elem[med_pos] = *(in_ptr + il + pixel_4frame);
            med_pos++;
        }
    }
}

*(out_ptr + il) = get_median (&med_elem[0], med_pos);
}

fprintf(stdout, "Done: MEDFIL\n");

return (1);
}

/*****
/*
/* Get median element (medianselector)
/*
/* Input:  Pointer to median elements      (med_ptr)
/*         Number of elements              (med_num)
/*
/* Returncode: Median element
/*
*****/

unsigned char  get_median (med_ptr, med_num)
unsigned char  *med_ptr;
int           med_num;
{
    unsigned char  med      = 0;
    int           i,
    ok            = FALSE;

    while (!ok)
    {
        /* Bubblesort */
        ok = TRUE;
        for (i = 0; i < med_num - 1; i++)
        {
            if (*(med_ptr + i) > *(med_ptr + i + 1))
            {
                /* Swap */

```

```

                                med
                                = *(med_ptr + i );
                                *(med_ptr + i ) = *(med_ptr + i + 1);
                                *(med_ptr + i + 1) = med;
                                ok = FALSE;
                                }
                                }
                                }
                                med = *(med_ptr + med_num / 2);

return (med);
}
-library_code_end

-library_mods
-library_mods_end

```

A.8 colplus

A.8.1 colplus.pane

```

-F 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +2+0 'COLORplus-Decoder' colplus
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input CVBS Signal' 'Input CVBS Signal' i1
-I 1 0 0 1 0 0 50x1+2+4 +0+0 ' ' 'Input Color Carrier' 'Input Color Carrier' i2
-I 1 0 1 0 0 0 50x1+2+6 +0+0 ' ' 'Input Impulse Response (Y low)' 'Impulse Response (Y low)' i3
-I 1 0 1 0 0 0 50x1+2+7 +0+0 ' ' 'Input Impulse Response (Y high)' 'Impulse Response (Y high)' i4
-I 1 0 1 0 0 0 50x1+2+8 +0+0 ' ' 'Input Impulse Response (C low)' 'Impulse Response (C low)' i5
-I 1 0 1 0 0 0 50x1+2+9 +0+0 ' ' 'Input Impulse Response (C high)' 'Impulse Response (C high)' i6
-O 1 0 0 1 0 0 50x1+2+11 +0+0 ' ' 'Output Y Component' 'Output Y Component' o1
-O 1 0 0 1 0 0 50x1+2+12 +0+0 ' ' 'Output U Component' 'Output U Component' o2
-O 1 0 0 1 0 0 50x1+2+13 +0+0 ' ' 'Output V Component' 'Output V Component' o3
-O 1 0 1 0 0 0 50x1+2+15 +0+0 ' ' 'Output Motion Detection Signal' 'Motion Detection Signal' o4
-T 1 0 0 1 0 40x1+2+17 +0+0 1 'Select COLORplus Decoding:' 'COLORplus Decoding' cp
-l 1 0 1 1 0 50x1+2+1 +0+0 0 'Simple COLORplus' 'False' 'True' 'Simple COLORplus' logical
-l 1 0 1 0 0 50x1+2+2 +0+0 0 'Full COLORplus' 'False' 'True' 'Full COLORplus' logical
-E
-b +2+21 'Parameters for Motion Detection:'
-i 1 0 1 1 0 50x1+4+22 +0+0 2 32 6 'X-Dimension of Detection Matrix' 'X-Dimension' xdim
-i 1 0 1 1 0 50x1+4+23 +0+0 2 32 6 'Y-Dimension of Detection Matrix' 'Y-Dimension' ydim
-i 1 0 1 1 0 50x1+4+24 +0+0 0 255 175 'Threshold for Hard Switching' 'Threshold' thresh
-T 1 0 0 1 0 40x1+2+26 +0+0 2 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 1 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-T 1 0 0 1 0 40x1+2+29 +0+0 1 'PAL Delay-Line:' 'PAL Delay-Line' dl
-l 1 0 1 1 0 1x1+2+1 +0+0 0 'Use PAL Delay-Line' 'False' 'True' 'Use PAL Delay-Line' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 'No PAL Delay-Line' 'False' 'True' 'No PAL Delay-Line' logical
-E
-R 1 0 1 10x2+1+32 'Execute' 'do operation' $PALPLUS/bin/colplus
-Q 1 0 10x2+20+32 'Quit'
-H 1 10x2+39+32 'Help' 'mam page for colplus' $PALPLUS/doc/manpages/colplus.1
-E
-E
-E

```

A.8.2 colplus.prog

```

-authors
Harald Zottmann
-authors_end

-short_prog_description

```

```

COLORplus-Decoder.
-short_prog_description_end

-short_lib_description
COLORplus-Decoder.
-short_lib_description_end

-man1_long_description
.I colplus
decodes a COLORplus-Signal.
.LP
A CVBS Signal coming from raw2cvbs will be
decoded as a COLORplus-Signal. Therefore it must
be connected with the first input. The second
input requires the color carrier which is provided
by raw2cvbs too. The last four inputs are reserved
for the required filters.
.LP
Description of filters:
.LP
Y low: low-pass filter for low frequency luminance.
.LP
Y high: high-pass filter for high frequency luminance. In
      COLORplus, the high frequency luminance is the same
      in every two fields of one frame.
.LP
C low: low-pass filter for demodulated u/v-components.
.LP
C high: high-pass filter for the chrominance.
.LP
The COLORplus-Decoder knows two different modes: 'Simple
COLORplus' and 'Full COLORplus'. In the simple-mode, the
decoder does COLORplus all the time. In the full-mode, an
additional motion detection is used. It detects fast moving
objects with a high color saturation, in order to switch to
the so called 'Fall-Back' mode. The motion detection is
performed, calculating and comparing the sum of a matrix in
the first and in the second field for the u- and the
v-component. The threshold selects the level of the
motion detection signal, for switching to the 'Fall-Back' mode,
which provides a high resolution in time for the color
components, but suppresses simultaneous the high frequency
luminance.
To visualize the signal for the motion detection,
it is provided at the optional output.
.LP
(Kays,R.: Ein Verfahren zur verbesserten PAL-Codierung
und -Decodierung. FKT, Vol. 44 (1990), No.11, pp 595-602)
.LP
(Ebner,A.; Matzel,E.; Morcom,R.; Ochs,R.; Riemann,U.;
Silverberg,M.; Storey,R.; Vreeswijk,F.; Westerkamp,D.:
PALplus: Uebertragung von 16:9-Bildern im terrestrischen
PAL-Kanal. FKT, Vol. 46 (1992), No.11, pp 733-739)
-man1_long_description_end

-man1_examples
colplus -i1 colorplus.dat -i2 carrier.dat
-i3 firfil_y_l -i4 firfil_y_h -i5 firfil_c_l -i6 firfil_c_h
-o1 y_out.dat -o2 u_out.dat -o3 v_out.dat
-cp 2 -xdim 6 -ydim 3 -thresh 120 -f_sc 1 -dl 2
.LP
Will decode the COLORplus-Signal in colorplus.dat, using the
color-carrier in carrier.dat. Other inputs contain impulse-
responses of fir-filters (length must be odd-numbered).
The output-signals contain yuv-components scaled according to
CCIR Rec.601. The decoder-mode is 'Full COLORplus'. The PAL
Delay-Line is switched off. The matrix for the motion-detection
has a dimension of 6x3 and the threshold for the hard switching
is 120.
-man1_examples_end

```

```

-man1_restrictions
.I colplus
.LP
Datatypes of input and output signals:
.LP
    input cvbs signal:      1_BYTE (0...255)
    input color carrier:    1_BYTE (0...255)
                           (128-127...128+127)
.LP
    input impulse response: FLOAT
.LP
    output signal y:        1_BYTE (16...235)
    output signal u:        1_BYTE (16...240)
                           (128-112...128+112)
    output signal v:        1_BYTE (16...240)
                           (128-112...128+112)
    output signal motion:   1_BYTE (0...255)
.LP
The length of the FIR-Filter must be odd-numbered.
-man1_restrictions_end

-man1_see_also
-man1_see_also_end

-man3_long_description
.I colplus
decodes a COLORplus-Signal.
.LP
A CVBS Signal coming from raw2cvbs will be
decoded as a COLORplus-Signal. Therefore it must
be connected with the first input. The second
input requires the color carrier which is provided
by raw2cvbs, too. The last four inputs are reserved
for the required filters.
.LP
Description of filters:
.LP
Y low: low-pass filter for low frequency luminance.
.LP
Y high: high-pass filter for high frequency luminance. In
        COLORplus, the high frequency luminance is the same
        in every two fields of one frame.
.LP
C low: low-pass filter for demodulated u/v-components.
.LP
C high: high-pass filter for the chrominance.
.LP
The COLORplus-Decoder knows two different modes: 'Simple
COLORplus' and 'Full COLORplus'. In the simple-mode, the
decoder does COLORplus all the time. In the full-mode, an
additional motion detection is used. It detects fast moving
objects with a high color saturation, in order to switch to
the so called 'Fall-Back' mode. The motion detection is
performed, calculating and comparing the sum of a matrix in
the first and in the second field for the u- and the
v-component. The threshold selects the level of the
motion detection signal, for switching to the 'Fall-Back' mode,
which provides a high resolution in time for the color
components, but suppresses simultaneous the high frequency
luminance.
To visualize the signal for the motion detection,
it is provided at the optional output.
-man3_long_description_end

-man3_restrictions
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

```

```

-usage_additions
-usage_additions_end

-include_includes
-include_includes_end

-include_additions
-include_additions_end

-include_macros
/*****
/* Read Input-Images */
*****/

#define READINPUT(cvbs_img, carrier_img); \
    cvbs_img = readimage(colplus->i1_file);\
    if(cvbs_img == NULL)\
    {\
        (void) fprintf(stderr,"colplus : cannot read input image");\
        (void) fprintf(stderr," %s\n",colplus->i1_file);\
        exit(1);\
    }\
    carrier_img = readimage(colplus->i2_file);\
    if(carrier_img == NULL)\
    {\
        (void) fprintf(stderr,"colplus : cannot read input image");\
        (void) fprintf(stderr," %s\n",colplus->i2_file);\
        exit(1);\
    }

#define READINPUT_FIL(fil_img, file_ptr); \
    fil_img = readimage(file_ptr);\
    if(fil_img == NULL)\
    {\
        (void) fprintf(stderr,"colplus : cannot read input image");\
        (void) fprintf(stderr," %s\n",file_ptr);\
        exit(1);\
    }

/*****
/* Check Input-Images */
*****/

#define CHECKINPUT(cvbs_img, carrier_img); \
    (void) proptype(program, cvbs_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, cvbs_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, cvbs_img, 1, TRUE);\
    (void) proper_map_enable(program, cvbs_img, VFF_MAP_OPTIONAL, TRUE);\
    (void) proptype(program, carrier_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, carrier_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, carrier_img, 1, TRUE);\
    (void) proper_map_enable(program, carrier_img, VFF_MAP_OPTIONAL, TRUE);\
    (void) matchsize (program, cvbs_img, carrier_img, TRUE);

#define CHECKINPUT_FIL(fil_img); \
    (void) proptype(program, fil_img, VFF_TYP_FLOAT, TRUE);\
    (void) proper_loc_type(program, fil_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, fil_img, 1, TRUE);\
    (void) proper_map_enable(program, fil_img, VFF_MAP_OPTIONAL, TRUE);
-include_macros_end

-main_variable_list
char      *file_ptr;

struct xvimage *cvbs_img,
               *carrier_img,
               *fil_img,
               *fil_yl_img,
               *fil_yh_img,

```

```

        *fil_cl_img,
        *fil_ch_img,
        *y_comp_img,
        *u_comp_img,
        *v_comp_img,
        *motion_img,
        *readimage();
-main_variable_list_end

-main_before_lib_call
    if (check_args()) exit(1);

    READINPUT (cvbs_img, carrier_img);
    CHECKINPUT (cvbs_img, carrier_img);

    if (colplus->i3_flag == 1)
    {
        file_ptr = colplus->i3_file;
        READINPUT_FIL (fil_img, file_ptr);
        CHECKINPUT_FIL (fil_img);
        fil_yl_img = fil_img;
    }

    if (colplus->i4_flag == 1)
    {
        file_ptr = colplus->i4_file;
        READINPUT_FIL (fil_img, file_ptr);
        CHECKINPUT_FIL (fil_img);
        fil_yh_img = fil_img;
    }

    if (colplus->i5_flag == 1)
    {
        file_ptr = colplus->i5_file;
        READINPUT_FIL (fil_img, file_ptr);
        CHECKINPUT_FIL (fil_img);
        fil_cl_img = fil_img;
    }

    if (colplus->i6_flag == 1)
    {
        file_ptr = colplus->i6_file;
        READINPUT_FIL (fil_img, file_ptr);
        CHECKINPUT_FIL (fil_img);
        fil_ch_img = fil_img;
    }
-main_before_lib_call_end

-main_library_call
    if(! lcolplus(cvbs_img, carrier_img,
        fil_yl_img, fil_yh_img, fil_cl_img, fil_ch_img,
        &y_comp_img, &u_comp_img, &v_comp_img, &motion_img,
        colplus->i3_flag, colplus->i4_flag,
        colplus->i5_flag, colplus->i6_flag,
        colplus->o4_flag,
        colplus->cp_toggle,
        colplus->xdim_int, colplus->xdim_flag,
        colplus->ydim_int, colplus->ydim_flag,
        colplus->thresh_int, colplus->thresh_flag,
        colplus->f_sc_toggle, colplus->dl_toggle) )
    {
        (void) fprintf(stderr, "lcolplus failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if(! writeimage(colplus->o1_file, y_comp_img))
    {
        (void) fprintf(stderr, "colplus : unable to write y output file\n");
    }

```

```

        exit(1);
    }
    if(! writeimage(colplus->o2_file, u_comp_img))
    {
        (void) fprintf(stderr,"colplus : unable to write u output file\n");
        exit(1);
    }
    if(! writeimage(colplus->o3_file, v_comp_img))
    {
        (void) fprintf(stderr,"colplus : unable to write v output file\n");
        exit(1);
    }
    if (colplus->o4_flag)
    {
        if(! writeimage(colplus->o4_file, motion_img))
        {
            (void) fprintf(stderr,"colplus : unable to write motion output file.\n");
            exit(1);
        }
    }
}
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lcolplus( cvbs_img,
              carrier_img,
              fil_yl_img,
              fil_yh_img,
              fil_cl_img,
              fil_ch_img,
              y_comp_img,
              u_comp_img,
              v_comp_img,
              motion_img,
              yl_flag,
              yh_flag,
              cl_flag,
              ch_flag,
              motion_flag,
              cp_toggle,
              xdim,
              xdim_flag,
              ydim,
              ydim_flag,
              thresh,
              thresh_flag,
              f_sc_toggle,
              dl_toggle)

struct xvimage *cvbs_img,
               *carrier_img,
               *fil_yl_img,
               *fil_yh_img,
               *fil_cl_img,
               *fil_ch_img,
               **y_comp_img,
               **u_comp_img,
               **v_comp_img,
               **motion_img;

int
               yl_flag,
               yh_flag,

```

```

        cl_flag,
        ch_flag,
        motion_flag,
        cp_toggle,
        xdim,
        xdim_flag,
        ydim,
        ydim_flag,
        thresh,
        thresh_flag,
        f_sc_toggle,
        dl_toggle;
-library_def_end

-library_code
/*****
/*
/* Library Code for colplus
/*
/*
*****/

#include "vinclude.h"
#include "math.h"

/*****/

#define FCARRIER      4433618.75
#define FROW           15625

#define SIMPLE_COLPLS  1
#define FULL_COLPLS    2

#define FC_FAC_4        1
#define FC_FAC_8        2
#define FC_FAC_12       3

#define DL_USE          1
#define DL_NO           2

#define LOW_PASS        1
#define HIGH_PASS       2

#define MAX_DIM         32          /* Matrix for Motion-Detection */

/*****/
/*
/* Main Function
/*
/*
*****/
{
    int          convo          ();
    int          field3         ();
    void         fieldadd       ();
    void         motiondetect   ();

    float        *fil_y1_ptr    = NULL, /* Pointer to FIR-Filter-Impulse-Response */
                *fil_yh_ptr    = NULL,
                *fil_cl_ptr    = NULL,
                *fil_ch_ptr    = NULL,
                mix             = 0.0,
                signal          = 0.0,
                delaysignal     = 0.0,
                carrier_fac     = 0.0,
                pixel_row       = 0.0;
    unsigned char *cvbs_ptr      = NULL, /* Pointer to Input-Image */
                *carrier_ptr    = NULL,
                *y_ptr          = NULL, /* Pointer to Output-Image */
                *u_ptr          = NULL,
                *v_ptr          = NULL,
                *m_ptr          = NULL,

```

```

signed long      *temp_ptr      = NULL;
                  il             = 0L,
                  cvbs_length    = 0L,
                  fil_yl_length  = 0L,
                  fil_yh_length  = 0L,
                  fil_cl_length  = 0L,
                  fil_ch_length  = 0L,
                  delay_line     = 0L,
                  byte_312row    = 0L,
                  byte_1frame    = 0L,
                  burstmode      = 0L,
                  deg_90         = 0L,
                  deg_180        = 0L;

/*****
/* Prepare Filters for use
*****/

fprintf(stdout, "\nCOLORplus-Decoder V2.0:\n");

switch (f_sc_toggle)
{
    case FC_FAC_4:
        carrier_fac = 4.0;
        break;

    case FC_FAC_8:
        carrier_fac = 8.0;
        break;

    case FC_FAC_12:
        carrier_fac = 12.0;
        break;

    default:
        fprintf (stderr, "Missing or improper argument in call to colplus.\n");
        return(0);
}

if (!xdim_flag) xdim = 1;
if (!ydim_flag) ydim = 1;

/*****

if (yl_flag)
{
    fil_yl_length = fil_yl_img->row_size * fil_yl_img->col_size;
    if (!(fil_yl_length % 2L))
    {
        fprintf (stderr, "Filter-Length (Y_l) must be odd-numbered.\n");
        return (0);
    }
    fil_yl_ptr = (float *) fil_yl_img->imagedata;
}
if (yh_flag)
{
    fil_yh_length = fil_yh_img->row_size * fil_yh_img->col_size;
    if (!(fil_yh_length % 2L))
    {
        fprintf (stderr, "Filter-Length (Y_h) must be odd-numbered.\n");
        return (0);
    }
    fil_yh_ptr = (float *) fil_yh_img->imagedata;
}
if (cl_flag)
{
    fil_cl_length = fil_cl_img->row_size * fil_cl_img->col_size;
    if (!(fil_cl_length % 2L))
    {
        fprintf (stderr, "Filter-Length (C_l) must be odd-numbered.\n");

```

```

        return (0);
    }
    fil_cl_ptr = (float *) fil_cl_img->imagedata;
}
if (ch_flag)
{
    fil_ch_length = fil_ch_img->row_size * fil_ch_img->col_size;
    if (!(fil_ch_length % 2L))
    {
        fprintf (stderr, "Filter-Length (C_h) must be odd-numbered.\n");
        return (0);
    }
    fil_ch_ptr = (float *) fil_ch_img->imagedata;
}

/*****

cvbs_ptr    = (unsigned char *) cvbs_img->imagedata;
carrier_ptr = (unsigned char *) carrier_img->imagedata;
cvbs_length = (signed long) (cvbs_img->row_size * cvbs_img->col_size);

*****/

/* delay_line = Number of pixels for PAL-Delay-Line (64.057 us)          */
/* 64.057 us = 64.000 us + 57.000 ns (284 Periods of Color-Carrier)      */
/*****

pixel_row   = (float)      ( carrier_fac * (FCARRIER / FROW           ));
delay_line  = (signed long) ( carrier_fac * (FCARRIER / FROW + 0.25 ));
byte_1frame = (signed long) ( carrier_fac * (FCARRIER / 25.0         ));
byte_312row = (signed long) ( carrier_fac * (FCARRIER / FROW * 312.00 )); + 1L;

*****/

/* Allocate memory for Output-Images                                     */
/*****

*y_comp_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
    1,1, "y_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
    VFF_LOC_IMPLICIT, 0);

*u_comp_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
    1,1, "u_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
    VFF_LOC_IMPLICIT, 0);

*v_comp_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
    1,1, "v_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
    VFF_LOC_IMPLICIT, 0);

*motion_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
    1,1, "motion_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
    VFF_LOC_IMPLICIT, 0);

if ((*y_comp_img == NULL) || (*u_comp_img == NULL) ||
    (*v_comp_img == NULL) || (*motion_img == NULL))
{
    if (*y_comp_img != NULL) freeimage(*y_comp_img);
    if (*u_comp_img != NULL) freeimage(*u_comp_img);
    if (*v_comp_img != NULL) freeimage(*v_comp_img);
    if (*motion_img != NULL) freeimage(*motion_img);
    fprintf (stderr, "Out of memory.\n");
    return (0);
}

y_ptr = (unsigned char *) (*y_comp_img)->imagedata;
u_ptr = (unsigned char *) (*u_comp_img)->imagedata;
v_ptr = (unsigned char *) (*v_comp_img)->imagedata;
m_ptr = (unsigned char *) (*motion_img)->imagedata;

temp_ptr = (unsigned char *)
    malloc((unsigned) cvbs_length * sizeof(char));
if (temp_ptr == NULL)

```

```

    {
        freeimage(*y_comp_img);
        freeimage(*u_comp_img);
        freeimage(*v_comp_img);
        freeimage(*motion_img);
        fprintf (stderr, "Out of memory.\n");
        return (0);
    }

/*****
/* Copy CVBS-Signal to Temporary-Memory */
*****/

    for (il = 0L; il < cvbs_length; il++)
    {
        *(temp_ptr + il) = *(cvbs_ptr + il);
    }

/*****
/* Convolution with Impulse-Response (Chrominance) */
*****/

    if (ch_flag)                /* High-Pass-Filter (C_h) */
    {
        fprintf(stdout, "Chrominance-Filter (High-Pass)\n");

        if (!convo (temp_ptr, fil_ch_ptr, cvbs_length, fil_ch_length, HIGH_PASS))
        {
            fprintf (stderr, "Convolution (C_h) failed.\n");
            return (0);
        }
    }

/*****
/* Demodulate U-Component */
/* Create U-Component-Outputimage */
*****/

    if (field3 (cvbs_ptr, carrier_fac) == 1)
    {
        fprintf (stdout, "Starting with field 3\n");
        burstmode = 1L;
    }
    else
    {
        fprintf (stdout, "Starting with field 1\n");
        burstmode = 0L;
    }

/*****
/* No PAL-Delay-Line (U-Component) */
*****/

    if (dl_toggle == DL_N0)
    {
        fprintf(stdout, "Demodulate U-Component (No PAL Delay-Line)\n");

        for (il = 0L; il < cvbs_length; il++)
        {
            signal = (float) *(temp_ptr + il) - 128.0;
            signal *= (float) *(carrier_ptr + il) - 128.0;
            *(u_ptr + il) = (unsigned char) (signal / 128.0 + 128.0);
        }
    }

/*****
/* Use PAL-Delay-Line (U-Component) */
*****/

    else

```

```

    {
        fprintf(stdout,"Demodulate U-Component (Use PAL Delay-Line)\n");

        for (il = 0L; il < cvbs_length; il++)
        {
            signal = (float) *(temp_ptr + il) - 128.0;
            if (il > delay_line)
            {
                delaysignal = (float) *(temp_ptr + il - delay_line) - 128.0;
            }

            else
            {
                delaysignal = 0.0;
            }

            signal += delaysignal;
            signal *= (float) *(carrier_ptr + il) - 128.0;
            *(u_ptr + il) = (unsigned char) (signal / 255.0 + 128.0);
        }
    }

    /******
    /* Demodulate V-Component
    /* Create V-Component-Outputimage
    /******

    deg_90 = (long) (carrier_fac / 4.0);
    deg_180 = (long) (carrier_fac / 2.0);
    carrier_ptr += deg_90;          /* +90 DEG */

    /******
    /* No PAL-Delay-Line (V-Component)
    /******

    if (dl_toggle == DL_NO)
    {
        fprintf(stdout,"Demodulate V-Component (No PAL Delay-Line)\n");

        for (il = 0L; il < cvbs_length; il++)
        {
            signal = (float) *(temp_ptr + il) - 128.0;
            if ((long) ((float) il / pixel_row) > burstmode)
            {
                burstmode++;
                if (burstmode % 2L)
                {
                    carrier_ptr -= deg_180;
                }
            }
            else
            {
                carrier_ptr += deg_180;
            }
        }
        signal *= (float) *(carrier_ptr + il) - 128.0;
        *(v_ptr + il) = (unsigned char) (signal / 128.0 + 128.0);
    }
}

    /******
    /* Use PAL-Delay-Line (V-Component)
    /******

    else
    {
        fprintf(stdout,"Demodulate V-Component (Use PAL Delay-Line)\n");

        for (il = 0L; il < cvbs_length; il++)
        {
            signal = (float) *(temp_ptr + il) - 128.0;
            if (il > delay_line)
            {

```

```

        delaysignal = (float) *(temp_ptr + il - delay_line) - 128.0;
    }
    else
    {
        delaysignal = 0.0;
    }
    if ((long) ((float) il / pixel_row) > burstmode)
    {
        burstmode++;
        if (burstmode % 2L)
        {
            carrier_ptr -= deg_180;
        }
        else
        {
            carrier_ptr += deg_180;
        }
    }
    signal -= delaysignal;
    signal *= (float) *(carrier_ptr + il) - 128.0;
    *(v_ptr + il) = (unsigned char) (signal / 255.0 + 128.0);
}

}

/*****
/* Free Carrier-Image to gain some more memory */
*****/

freeimage (carrier_img);

/*****
/* Convolution with Impulse-Response (U/V-Component) */
*****/

if (cl_flag)                /* Low-Pass-Filter (C_l) */
{
    fprintf(stdout,"U/V-Filter (Low-Pass)\n");

    if (!convo (u_ptr, fil_cl_ptr, cvbs_length, fil_cl_length, LOW_PASS))
    {
        fprintf (stderr, "Convolution (C_l) failed.\n");
        return (0);
    }

    if (!convo (v_ptr, fil_cl_ptr, cvbs_length, fil_cl_length, LOW_PASS))
    {
        fprintf (stderr, "Convolution (C_l) failed.\n");
        return (0);
    }
}

}

/*****
/* Full Color-Plus (Motion-Detection) */
/* Get Motion-Detection-Signal from U/V-Components */
*****/

if (cp_toggle == FULL_COLPLS)                /* Full Color-Plus */
{
    fprintf(stdout,"Full COLORplus: Motion-Detection\n");

    motiondetect (u_ptr, v_ptr, m_ptr, cvbs_length,
                  xdim, ydim, thresh, thresh_flag, carrier_fac);

/*****
/* Store demodulated U/V-Components in temporary-memory */
*****/

    for (il = 0L; il < cvbs_length; il++)
    {
        *(temp_ptr + il) = *(u_ptr + il);
        *(y_ptr + il) = *(v_ptr + il);
    }
}

```

```

    }
}
else /* Simple Color-Plus */
{
    for (il = 0L; il < cvbs_length; il++)
    {
        *(m_ptr + il) = (unsigned char) 0;
    }
}

/*****

fprintf(stdout, "Field-Adding-Device (U/V)\n");

fieldadd (u_ptr, cvbs_length, byte_312row, byte_1frame);
fieldadd (v_ptr, cvbs_length, byte_312row, byte_1frame);

*****/
/* Full Color-Plus (U/V-Components) */
/* Mix Signals according to Motion-Detection-Signal */
/*****/

if (cp_toggle == FULL_COLPLS) /* Full Color-Plus */
{
    fprintf(stdout, "Full COLORplus: Mixer (U/V-Components)\n");

    for (il = 0L; il < cvbs_length; il++) /* Color-Plus / Fall-Back-Modus */
    {
        mix = (float) *(m_ptr + il) / 255.0;

        *(u_ptr + il) = (unsigned char)
            ( (float) *(u_ptr + il) * (1.0 - mix)
              + (float) *(temp_ptr + il) * (mix) );
        *(v_ptr + il) = (unsigned char)
            ( (float) *(v_ptr + il) * (1.0 - mix)
              + (float) *(y_ptr + il) * (mix) );
    }
}

/*****/
/* Copy CVBS-Signal to Y-Component and to temporary-memory */
/*****/

for (il = 0L; il < cvbs_length; il++)
{
    *(y_ptr + il) = *(cvbs_ptr + il);
    *(temp_ptr + il) = *(cvbs_ptr + il);
}

/*****/
/* Free CVBS-Image to gain some more memory */
/*****/

freeimage (cvbs_img);

/*****/
/* Convolution with Impulse-Response (Luminance) */
/*****/

if (yl_flag) /* Low-Pass-Filter (Y_l) */
{
    fprintf(stdout, "Luminance-Filter (Low-Pass)\n");

    if (!convo (y_ptr, fil_yl_ptr, cvbs_length, fil_yl_length, LOW_PASS))
    {
        fprintf (stderr, "Convolution (Y_l) failed.\n");
        return (0);
    }
}

```

```

if (yh_flag)                /* High-Pass-Filter (Y_h) */
{
    fprintf(stdout,"Luminance-Filter (High-Pass)\n");

    if (!convo (temp_ptr, fil_yh_ptr, cvbs_length, fil_yh_length, HIGH_PASS))
    {
        fprintf (stderr, "Convolution (Y_h) failed.\n");
        return (0);
    }
}

fprintf(stdout,"Field-Adding-Device (Y)\n");

fieldadd (temp_ptr, cvbs_length, byte_312row, byte_1frame);

/*****
/* Full Color-Plus (Y-Component)
/* Mix Signals according to Motion-Detection-Signal
*****/

if (cp_toggle == FULL_COLPLS)    /* Full Color-Plus */
{
    fprintf(stdout,"Full COLORplus: Mixer (Y-Component)\n");

    for (il = 0L; il < cvbs_length; il++)    /* Color-Plus / Fall-Back-Modus */
    {
        mix = (float) *(m_ptr + il) / 255.0;

        *(y_ptr + il) = (unsigned char)
            ( ((float) *(y_ptr + il)
              + ((float) *(temp_ptr + il) - 128.0) * (1.0 - mix) ));
    }
}
else                                /* Simple Color-Plus */
{
    fprintf(stdout,"Simple COLORplus: Add Y-High and Y-Low\n");

    for (il = 0L; il < cvbs_length; il++)
    {
        *(y_ptr + il) = (unsigned char)
            ( ((float) *(y_ptr + il)
              + ((float) *(temp_ptr + il) - 128.0) ));
    }
}

free (temp_ptr);

/*****
/* Scale YUV-Components according to CCIR-Rec.601
*****/

fprintf(stdout,"Scale YUV according to CCIR-Rec.601\n");

for (il = 0L; il < cvbs_length; il++)
{
    signal = (float) *(y_ptr + il);                /* Y-Component */
    signal = (signal / 143.0 * 219.0) - 76.0;
    if (signal > 235.0) signal = 235.0;
    if (signal < 0.0) signal = 0.0;
    *(y_ptr + il) = (unsigned char) signal;

    signal = (float) *(u_ptr + il);                /* U-Component */
    signal = (signal - 127.0) / 31.0;
    if (signal > 1.0) signal = 1.0;
    if (signal < -1.0) signal = -1.0;
    *(u_ptr + il) = (unsigned char) (signal * 112.0) + 128.0;

    signal = (float) *(v_ptr + il);                /* V-Component */
    signal = (signal - 127.0) / 44.0;
    if (signal > 1.0) signal = 1.0;

```

```

        if (signal < -1.0) signal = -1.0;
        *(v_ptr + il) = (unsigned char) (signal * 112.0) + 128.0;
    }

    fprintf(stdout, "Done: COLORplus-Decoder\n");

return (1);
}

/*****
/*
/* Motion-Detection in U/V-Components
/*
/* Input:  Pointer to U-Inputsignal          (u_in)
/*         Pointer to V-Inputsignal          (v_in)
/*         Length of Inputsignals            (length)
/*         X-Dimension of Matrix             (xdim)
/*         Y-Dimension of Matrix             (ydim)
/*         Threshold for Hard Switching      (thresh / thresh_flag)
/*
/* Output: Pointer to Outputsignal           (m_out)
/*
*****/

void motiondetect (u_in, v_in, m_out, length, xdim, ydim, thresh,
thresh_flag, carrier_fac)
unsigned char *u_in,
*v_in,
*m_out;
signed long length;
int xdim,
ydim,
thresh,
thresh_flag;
float carrier_fac;
{
unsigned char out;
float u_sig_1 = 0.0,
u_sig_2 = 0.0,
v_sig_1 = 0.0,
v_sig_2 = 0.0,
u_diff = 0.0,
v_diff = 0.0,
signal = 0.0;
signed long il, kl,
xl, yl,
xoffset,
yoffset,
foffset,
offset,
count,
len_row[MAX_DIM],
len_312row = 0L,
len_1frame = 0L,
num_frames = 0L;

len_1frame = (signed long) ( carrier_fac * (FCARRIER / 25.0 ));
len_312row = (signed long) ( carrier_fac * (FCARRIER / FROW * 312.00 )) + 1L;
num_frames = (signed long) ( length / len_1frame );

for (il = 0L; il < MAX_DIM; il++) /* Init rowlength variables */
{
    len_row[il] = (signed long) (carrier_fac * (FCARRIER / FROW)) * il;
}

/*****
/*
/* Motion-Detection */
*****/

for (il = 0L; il < num_frames; il++)
{
    foffset = (signed long) (il * len_1frame);

```

```

xoffset = 0L; yoffset = 0L;

while (yoffset < 312L)
{
    offset = foffset + xoffset + (yoffset * len_row[1L]);
    u_sig_1 = 0.0; u_sig_2 = 0.0;
    v_sig_1 = 0.0; v_sig_2 = 0.0;
    count = 0L;

    for (yl = 0L; yl < ydim; yl++)
    {
        if ((yl + yoffset) >= 312L)
        {
            continue;
        }
        for (xl = 0L; xl < xdim; xl++)
        {
            if ((xl + xoffset) >= len_row[1L])
            {
                continue;
            }
            if ((yl + yoffset) >= 312L)
            {
                continue;
            }
            count++;
            u_sig_1 += (float) u_in[ offset + xl + len_row[yl] ];
            v_sig_1 += (float) v_in[ offset + xl + len_row[yl] ];
            u_sig_2 += (float) u_in[ offset + xl + len_row[yl] + len_312row ];
            v_sig_2 += (float) v_in[ offset + xl + len_row[yl] + len_312row ];
        }
    }

    /*****

    u_diff = (float) fabs ((double) u_sig_1 - (double) u_sig_2);
    if (count > 0L) u_diff /= (float) count;
    u_diff /= 62.0;

    v_diff = (float) fabs ((double) v_sig_1 - (double) v_sig_2);
    if (count > 0L) v_diff /= (float) count;
    v_diff /= 88.0;

    signal = (u_diff + v_diff) * 512.0;          /* Scale */
    if (signal > 255.0)    signal = 255.0;      /* Cut */
    if (signal < 0.0)     signal = 0.0;

    if (thresh_flag)
    {
        if (signal > thresh)    out = 255;
        else                   out = 0;
    }
    else
    {
        out = (unsigned char) signal;
    }

    *****/

    for (yl = 0L; yl < ydim; yl++) /* Init Detection Signal */
    {
        if ((yl + yoffset) >= 312L)
        {
            continue;
        }
        for (xl = 0L; xl < xdim; xl++)
        {
            if ((xl + xoffset) >= len_row[1L])
            {

```

```

        continue;
    }
    m_out[ offset + xl + len_row[yl] ] = out;
    m_out[ offset + xl + len_row[yl] + len_312row ] = out;
}

}

/*****
    xoffset += xdim;
    if (xoffset > len_row[1L])          /* Right Border */
    {
        xoffset = 0L;
        yoffset += ydim;
    }
} /* while */
} /* for */

return;
}

/*****
/*
/* Add two fields (Color-Plus)
/*
/* Input:  Pointer to Inputsignal to be processed (datain)
/*          Length of Inputsignal to be processed (dpoints)
/*          Length of 312 rows (len_312row)
/*          Length of one frame (len_1frame)
/*
/* Output: Pointer to Outputsignal (datain)
/*
/*
*****/

void fieldadd (datain, dpoints, len_312row, len_1frame)
signed long    dpoints,
               len_312row,
               len_1frame;
unsigned char  *datain;
{
    float      signal      = 0.0;
    signed long il, kl,
               offset      = 0L,
               num_frames  = 0L;

    num_frames = dpoints / len_1frame;

    for (il = 1L; il <= num_frames; il++)
    {
        offset = (signed long) (il * len_1frame) - 1L;

        for (kl = 0L; kl < len_312row; kl++)
        {
            signal = ( 0.5 * (float) datain[offset - kl]
                      + 0.5 * (float) datain[offset - len_312row - kl]);
            datain[offset - kl] = (unsigned char) signal;
            datain[offset - len_312row - kl] = (unsigned char) signal;
        }
    }

    return;
}

/*****
/*
/* Performs a circular convolution of 1_BYTE-Signals (unsigned char)
/* with an Impulse-Response (float).
/*
/*
/* Input:  Data vector of Color-Component (datain1)
/*          Data vector of Impulse-Response (datain2)
/*          Number of data points (dpoints1, dpoints2)
*****/

```

```

/*          Filtermode (LOW_PASS / HIGH_PASS)  (mode)                      */
/*                                                                 */
/* Output: Resultant convolution data          (datain1)              */
/*                                                                 */
/*                                                                 */
/*****

int      convo (datain1, datain2, dpoints1, dpoints2, mode)
signed long  dpoints1,
             dpoints2;
unsigned char *datain1;
float        *datain2;
int          mode;
{
signed long  shift,
             idx,
             n, h;
float        *conv,
             temp;

    if (dpoints1 < dpoints2)
    {
        fprintf(stderr,"Data vector 1 must be longer than Data vector 2\n");
        return (0);
    }

    conv = (float *) malloc((unsigned)(dpoints1 * sizeof(float)));
    if (conv == NULL)
    {
        fprintf(stderr,"Out of memory.\n");
        return (0);
    }

    for (n = 0L; n < dpoints1; n++)
    {
        temp = 0.0;
        for (h = 0L; h < dpoints2; h++)
        {
            idx = n - h;
            if (idx < 0L)
            {
                idx += dpoints1;
            }
            temp += datain2[h] * (float) datain1[idx];
        }
        conv[n] = temp;
    }

    if (mode == LOW_PASS)
    {
        for (n = 0L; n < dpoints1; n++)
        {
            temp = conv[n];
            if (temp > 255.0)
            {
                temp = 255.0;
            }
            if (temp < 0.0)
            {
                temp = 0.0;
            }
            datain1[n] = (unsigned char) temp;
        }
    }
    else if (mode == HIGH_PASS)
    {
        for (n = 0L; n < dpoints1; n++)
        {
            temp = conv[n];
            temp += 128.0;
            if (temp > 255.0)

```

```

        {
            temp = 255.0;
        }
        if (temp < 0.0)
        {
            temp = 0.0;
        }
        datain1[n] = (unsigned char) temp;
    }

    else
    {
        fprintf(stderr, "Unknown type of filter.\n");
        free (conv);
        return (0);
    }

    shift = dpoints2 / 2;
    for (n = 0L; n < dpoints1 - shift; n++)
    {
        datain1[n] = datain1[n + shift];
    }

    free (conv);

return (1);
}

/*****
/*
/* Checks for burst in field 3
/*
/* Input:  Pointer (ptr) to CVBS-Signal
/*          Samplefrequency (fsc_fac * Frequency of Color-Carrier)
/*
/* Returncode: 1 if burst found (--> field 3) - 0 if burst not found (--> field 1)
/*
*****/

int      field3 (ptr, fsc_fac)
unsigned char *ptr;
double      fsc_fac;
{
    signed long    burst_6_offset = 0L,
                    burst_9_offset = 0L,
                    burst_length  = 0L,
                    mean_start    = 0L,
                    mean_end      = 0L,
                    il;
    float          pixel_row      = 0.0,
                    mean_level    = 0.0,
                    val           = 0.0,
                    sum_6         = 0.0,
                    sum_9         = 0.0;

    pixel_row      = (float)      (FCARRIER / FROW * fsc_fac);
    burst_6_offset = (signed long) ( 5.0 * pixel_row) + (long) ( 25 * fsc_fac);
    burst_9_offset = (signed long) ( 8.0 * pixel_row) + (long) ( 25 * fsc_fac);
    mean_start     = (signed long) ( 5.0 * pixel_row) + (long) ( 50 * fsc_fac);
    mean_end       = (signed long) ( 5.0 * pixel_row) + (long) (250 * fsc_fac);
    burst_length   = (signed long) ( 8.0 * fsc_fac);

    for (il = mean_start; il < mean_end; il++)
    {
        mean_level += (float) *(ptr + il);
    }
    mean_level /= (float) (mean_end - mean_start);

    for (il = 0L; il < burst_length; il++) /* Burstamplitude row 6 is ??? % */
    {

```

```

        val    = (float) *(ptr + burst_6_offset + il);
        val    -= (float) mean_level;
        sum_6 += (float) abs ((int) val);
    }
    for (il = 0L; il < burst_length; il++) /* Burstamplitude row 9 is 100 % */
    {
        val    = (float) *(ptr + burst_9_offset + il);
        val    -= (float) mean_level;
        sum_9 += (float) abs ((int) val);
    }

    if (sum_9 > 0.0)
    {
        val = sum_6 / sum_9;
    }
    else
    {
        return (-1);          /* Error */
    }

    if (val > 0.9)             /* 90 % */
    {
        return (1);           /* Field 3 */
    }

    return (0);                /* Field 1 */
}
-library_code_end

-library_mods
-library_mods_end

```

A.9 line23

A.9.1 line23.pane

```

-F 4.2 1 0 170x7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +0+0 'Detect and Decode PALplus Signalling Bits' line23
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input CVBS Signal' 'Input CVBS Signal' i
-O 1 0 1 0 0 0 50x1+2+4 +0+0 ' ' 'Output Data Signal' 'Output Data Signal' o1
-O 1 0 1 0 0 0 50x1+2+6 +0+0 ' ' 'Information (ASCII)' 'Information (ASCII)' o2
-l 1 0 0 1 0 50x1+2+8 +0+0 1 'CRC-8 check bits error detection' 'OFF' 'ON' 'Error' error_detect
-l 1 0 0 1 0 50x1+2+9 +0+0 1 'Full ASCII Information' 'OFF' 'ON' 'Full ASCII' full_ascii
-T 1 0 0 1 0 40x1+2+11 +0+0 2 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 1 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-R 1 0 1 10x2+1+14 'Execute' 'do operation' $PALPLUS/bin/line23
-Q 1 0 10x2+20+14 'Quit'
-H 1 10x2+39+14 'Help' 'man page for line23' $PALPLUS/doc/manpages/line23.1
-E
-E
-E

```

A.9.2 line23.prog

```

-authors
Harald Zottmann
-authors_end

-short_prog_description
Gets the data bits from line 23.

```

```

-short_prog_description_end

-short_lib_description
Gets the data bits from line 23.
-short_lib_description_end

-man1_long_description
.I line23
gets the data bits from line 23.
.LP
The data bits in the CVBS Signal (PALplus), generated from
raw2cvbs will be demodulated. The ASCII Information will
be sent to standard output or to the ASCII File Viewer
(xvviewer) if it is connected to the second output.
.LP
The creation of an output data signal is optional. The
data signal contains the 32 data bits (b00-b31) and the
WSL bit (Wide Screen Letterbox). If no data bits were found,
all bytes in the data signal are set to zero.
.LP
Optional error detection is performed by using the CRC-8
polynomial. If the CRC-8 check bits are unset, error
detection is switched off.
.LP
By selecting the full ascii information button, all information
about error detection (if selected) and bi-phase elements is
provided.
.LP
The format of the data output image is:
.LP
b0 b1 b2...b29 b30 b31 WSL (-> 33 * 1_BYTE).
-man1_long_description_end

-man1_examples
line23 -i cvbs.dat -o1 data_bits.dat -f_sc 1
.LP
will generate data_bits.dat from cvbs.dat.
-man1_examples_end

-man1_restrictions
.I line23
will only operate with CVBS Signals of data storage
type 1_BYTE.
-man1_restrictions_end

-man1_see_also
-man1_see_also_end

-man3_long_description
.I line23
gets the data bits from line 23.
.LP
The data bits in the CVBS Signal (PALplus), generated from
raw2cvbs will be demodulated. The ASCII Information will
be sent to standard output or to the ASCII File Viewer
(xvviewer) if it is connected to the second output.
.LP
The creation of an output data signal is optional. The
data signal contains the 32 data bits (b00-b31) and the
WSL bit (Wide Screen Letterbox). If no data bits were found,
all bytes in the data signal are set to zero.
.LP
Optional error detection is performed by using the CRC-8
polynomial. If the CRC-8 check bits are unset, error
detection is switched off.
.LP
By selecting the full ascii information button, all information
about error detection (if selected) and bi-phase elements is
provided.
.LP

```

```

The format of the data output image is:
.LP
b0 b1 b2...b29 b30 b31 WSL (-> 33 * 1_BYTE).
-man3_long_description_end

-man3_restrictions
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

-usage_additions
-usage_additions_end

-include_includes
-include_includes_end

-include_additions
-include_additions_end

-include_macros
/*****
/* Read Input-Image
*****/

#define READINPUT(cvbs_img); \
    cvbs_img = readimage(line23->i_file);\
    if(cvbs_img == NULL)\
    {\
        (void) fprintf(stderr,"line23 : cannot read input image");\
        (void) fprintf(stderr," %s\n",line23->i_file);\
        exit(1);\
    }

/*****
/* Check Input-Images
*****/

#define CHECKINPUT(cvbs_img); \
    (void) proptype(program, cvbs_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, cvbs_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, cvbs_img, 1, TRUE);\
    (void) proper_map_enable(program, cvbs_img, VFF_MAP_OPTIONAL, TRUE);
-include_macros_end

-main_variable_list
struct xvimage  *cvbs_img,
                *data_img,
                *readimage();

FILE            *write_device;
-main_variable_list_end

-main_before_lib_call
    if (check_args()) exit(1);

    READINPUT  (cvbs_img);
    CHECKINPUT (cvbs_img);

    if (line23->o2_flag)
    {
        if (! vwrite_ascii(line23->o2_file, &write_device))
        {
            (void) fprintf(stderr, "%s: Can't open ascii output file\n",program);
            exit(1);
        }
    }
    else
    {
        write_device = stdout;

```

```

    }
-main_before_lib_call_end

-main_library_call
    if(!lline23(  cvbs_img,
                  &data_img,
                  write_device,
                  line23->o1_flag, line23->o2_flag,
                  line23->error_detect_logic,
                  line23->full_ascii_logic,
                  line23->f_sc_toggle))
    {
        (void) fprintf(stderr, "lline23 failed.\n");
        exit(1);
    }
-main_library_call_end

-main_after_lib_call
    if (line23->o1_flag)
    {
        if(! writeimage(line23->o1_file, data_img))
        {
            (void) fprintf(stderr, "line23 : unable to write data output file.\n");
            exit(1);
        }
    }
    if (line23->o2_flag)
    {
        fclose(write_device);
    }
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lline23 (  cvbs_img, data_img,
              write_device,
              data_flag,
              ascii_flag,
              error_detect,
              full_ascii,
              f_sc_toggle)

struct xvimage  *cvbs_img,
                **data_img;

FILE            *write_device;

int             data_flag,
                ascii_flag,
                error_detect,
                f_sc_toggle;
-library_def_end

-library_code
/*****
/*
/* Library Code for line23
/*
/*
*****/

#include "vinclude.h"

```

```

/*****/

#define DCARRIER      5000000.00
#define FCARRIER      4433618.75
#define FROW           15625.00

#define DATA_THRESH   128
#define NUM_ALL_BITS   32+10+16+10+32+10+16+16

#define FC_FAC_4        1
#define FC_FAC_8        2
#define FC_FAC_12       3

#define OK_DATA          "Data-Bits found in field: %ld\n"
#define NO_DATA          "No Data-Bits found in field: %ld\n"
#define NEW_DATA         "New Data-Bits in field: %ld\n"
#define PREAMBEL         "Preamble in field: %ld is incorrect\n"
#define WARNING_WSL      "WARNING: WSL-Bit has alternated in field: %ld\n"
#define ERROR_00_11      "ERROR: 00 or 11 found in Data-Group of field: %ld\n"
#define ERROR_CRC_8      "ERROR: CRC-8 detected an error in field: %ld\n"

/*****/
/*
/* Main Function
/*
/*****/

{
int    getdatabits      ();
int    getws1           ();
int    getdatagroups    ();
int    errordetection   ();
void    databits        ();
void    printline       ();

unsigned char    *cvbs_ptr      = NULL,
                 *data_ptr      = NULL,
                 all_bits1      [NUM_ALL_BITS],
                 all_bits2      [NUM_ALL_BITS],
                 data_groups     [40],
                 preamble       [32] = {1,1,1,0,    1,0,1,0,    1,0,1,0,    1,0,1,0,
                                     1,1,1,0,    1,0,0,1,    0,1,0,0,    0,1,1,0};

signed long      field          = 0L,
                 num_fields     = 0L,
                 total_bytes    = 0L,
                 field_bytes    = 0L,
                 row_bytes      = 0L;

double           carrier_fac    = 0.0,
                 bytes_per_bit  = 0.0,
                 samp_freq      = 0.0;

int              i,
                 done_this_bits = FALSE,
                 new_wsl        = -1,
                 old_wsl        = -1;

/*****/

switch (f_sc_toggle)
{
case FC_FAC_4:
carrier_fac = 4.0;
break;

case FC_FAC_8:
carrier_fac = 8.0;
break;

case FC_FAC_12:
carrier_fac = 12.0;
break;
}

```

```

        default:
        fprintf (stderr,"Missing or improper argument in call to line23.\n");
        return (0);
    }
    samp_freq = FCARRIER * carrier_fac;

    /******
    /* Duration of one field 1/50 sec = 0.02 sec
    /******

    field_bytes      = (long)   (0.02 * samp_freq);
    total_bytes      = (long)   (cvbs_img->col_size * cvbs_img->row_size);
    row_bytes        = (long)   (FCARRIER / FROW * carrier_fac);
    num_fields       = (long)   (total_bytes / field_bytes);
    bytes_per_bit    = (double) (samp_freq / DCARRIER);

    /******
    /* Allocate memory for the output image
    /*
    /* Format of output image: b00-b31 / WSL-Bit (-> 33 Bytes)
    /******

    if (data_flag)
    {
        *data_img = createimage(33,1,VFF_TYP_1_BYTE,1,1,
                               "created by line23",0,0,VFF_MS_NONE,
                               VFF_MAPTYP_NONE,VFF_LOC_IMPLICIT,0);
        if(*data_img == NULL)
        {
            fprintf(stderr,"lline23 : unable to allocate memory.\n");
            return (0);
        }
    }

    /******
    /* Cast to the proper data type
    /******

    cvbs_ptr = (unsigned char *) cvbs_img->imagedata;
    if (data_flag)
    {
        data_ptr = (unsigned char *) (*data_img)->imagedata;
        memset (data_ptr, 0x00, 33);
    }

    /******

    fprintf (stdout,"\nLINE23 V1.2:\n");

    fprintf (write_device,"Number of fields:      %ld\n", num_fields);
    fprintf (write_device,"Samplefrequency (MHz):  %f\n",  samp_freq / 1000000.0);
    switch (error_detect)
    {
        case 0:
            fprintf (write_device,"CRC-8 Error Detection:  NO\n");
            break;

        default:
            fprintf (write_device,"CRC-8 Error Detection:  YES\n");
            break;
    }
    switch (full_ascii)
    {
        case 0:
            fprintf (write_device,"Full ASCII-Information:  NO\n");
            break;

        default:
            fprintf (write_device,"Full ASCII-Information:  YES\n");

```

```

        break;
    }
    switch (data_flag)
    {
        case 0:
            fprintf (write_device,"Create Data-Signal:      NO\n");
            break;

        default:
            fprintf (write_device,"Create Data Signal:      YES\n");
            break;
    }

/*****

    printline (write_device);

    for (field = 0; field < num_fields; field++)
    {
        if (! getdatabits (cvbs_ptr, &all_bits1[0], field, field_bytes,
                           row_bytes, bytes_per_bit))
        {
            if (!(field % 2L))          /* Data-Bits missing in field 0,2,4... */
            {
                fprintf (write_device, NO_DATA, field);
                printline (write_device);
            }
        }
        else
        {
            fprintf (write_device, OK_DATA, field);
            if (strcmp (&all_bits1[0], &preamble[0], 32) != 0)
            {
                fprintf (write_device, PREAMBEL, field);
                printline (write_device);
                continue;
            }
            new_wsl = getwsl (&all_bits1[0]);
            if (new_wsl != old_wsl)
            {
                if (old_wsl != -1)
                    fprintf (write_device,WARNING_WSL, field);
                old_wsl = new_wsl;
            }
            if (! getdatagroups (&data_groups[0], &all_bits1[0]))
            {
                fprintf(write_device,ERROR_00_11, field);
                printline (write_device);
                continue;
            }
        }
    }

/*****

        if (error_detect)          /* Error detection is switched on */
        {
            if (! errordetection (&data_groups[0], write_device, full_ascii))
            {
                fprintf (write_device,ERROR_CRC_8, field);
                printline (write_device);
                continue;
            }
        }

/*****

        if (!done_this_bits)
        {
            memcpy (&all_bits2[0], &all_bits1[0], NUM_ALL_BITS);
        }
        if (strcmp (&all_bits1[0], &all_bits2[0], NUM_ALL_BITS) != 0)

```

```

        {
            fprintf (write_device, NEW_DATA, field);
            done_this_bits = FALSE;
        }

/*****
/* Write Data-Bits to Output-Image
*****/

        if (data_flag && !done_this_bits)
        {
            memcpy (data_ptr, &data_groups[0], 32);
            *(data_ptr + 32) = (unsigned char) new_wsl;
        }

/*****
/* Write Data-Bits to standard-output
*****/

        if (full_ascii || !done_this_bits)
        {
            fprintf (write_device, "\nWSL: %d", (int) new_wsl);
            fprintf (write_device, "\nData-Group 1 (b00...b07): ");
            for (i = 0; i < 8; i++) fprintf(write_device, "%d", (int) data_groups[i]);
            fprintf (write_device, "\nData-Group 2 (b08...b15): ");
            for (i = 8; i < 16; i++) fprintf(write_device, "%d", (int) data_groups[i]);
            fprintf (write_device, "\nData-Group 3 (b16...b23): ");
            for (i = 16; i < 24; i++) fprintf(write_device, "%d", (int) data_groups[i]);
            fprintf (write_device, "\nData-Group 4 (b24...b31): ");
            for (i = 24; i < 32; i++) fprintf(write_device, "%d", (int) data_groups[i]);
            fprintf (write_device, "\nCRC-8-Bits (b32...b39): ");
            for (i = 32; i < 40; i++) fprintf(write_device, "%d", (int) data_groups[i]);

            databits (&data_groups[0], new_wsl, write_device);

            done_this_bits = TRUE;
        }

        if (full_ascii)
        {
            fprintf (write_device, "\nNotation in bi-phase elements\n");
            fprintf (write_device, "Preamble: ");
            for (i = 0; i < 32; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\nWSL: ");
            for (i = 32; i < 42; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\nData-Group 1: ");
            for (i = 42; i < 58; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\nWSL: ");
            for (i = 58; i < 68; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\nData-Group 2: ");
            for (i = 68; i < 84; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\nData-Group 3: ");
            for (i = 84; i < 100; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\nWSL: ");
            for (i = 100; i < 110; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\nData-Group 4: ");
            for (i = 110; i < 126; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\nCRC-8-Bits: ");
            for (i = 126; i < 142; i++) fprintf(write_device, "%d", (int) all_bits1[i]);
            fprintf (write_device, "\n");
        }

        printline (write_device);
    }

    fprintf (stdout, "Done: LINE23\n\n");

return (1);
}

```

```

/*****
void    printline (write_device)
FILE    *write_device;
{
    fprintf (write_device,"-----\n");
return;
}

/*****
/*
/* Gets all Data-Bits in line 23 of specified field
/*
/* Input:  Pointer to CVBS-Signal (PALplus)
/*         Pointer to storage area for Data-Bits
/*         Number of field to search
/*         Bytes per field
/*         Bytes per row
/*         Bytes per Bit (Number of Bytes in CVBS-Signal for one Bit)
/*
/* Output: Data-Bits
/*
/* Returncode: 1 if Data-Bits found - 0 if Data-Bits not found
/*
*****/

int      getdatabits (cvbs_ptr, bit_ptr, field, field_bytes, row_bytes, bytes_per_bit)
unsigned char  *cvbs_ptr,
               *bit_ptr;
signed long    field,
               field_bytes,
               row_bytes;
double         bytes_per_bit;
{
signed long    num_of_bits    ();

signed int     high    = 1,
               num     = 0;
signed long    i       = 0L,
               j       = 0L,
               k       = 0L,
               count    = 0L,
               line_23 = 0L;

memset (bit_ptr, 0x00, NUM_ALL_BITS);

line_23 = (field * field_bytes) + (row_bytes * 22);

while ((j < row_bytes) && (*(cvbs_ptr + line_23 + j) < DATA_THRESH))
{
    j++;
}
if (j >= row_bytes - 1L)
{
    return (0);
}

for (i = j; i < row_bytes; i++)
{
    if (*(cvbs_ptr + line_23 + i) > DATA_THRESH)
    {
        if (high == 1)
        {
            count++;
        }
        else
        {
            for (k = 0; k < num_of_bits(count, bytes_per_bit); k++)
            {
                *(bit_ptr + num) = 0;
            }
        }
    }
}
}

```

```

        num++;
        if (num > NUM_ALL_BITS)
        {
            return (0);
        }
        high = 1;
        count = 1;
    }
else
{
    if (high == 0)
    {
        count++;
    }
    else
    {
        for (k = 0; k < num_of_bits(count, bytes_per_bit); k++)
        {
            *(bit_ptr + num) = 1;
            num++;
            if (num > NUM_ALL_BITS)
            {
                return (0);
            }
        }
        high = 0;
        count = 1;
    }
}

if (num < NUM_ALL_BITS - 1)
{
    return (0);
}

return (1);
}

/*****/

signed long    num_of_bits (bytes, bytes_per_bit)
signed long    bytes;
double         bytes_per_bit;
{
    signed long    i_num = 0L;
    double         d_num = 0.0;

    d_num = (double) bytes / bytes_per_bit;
    i_num = (signed long) d_num;
    d_num -= (double) i_num;
    if (d_num >= 0.50)
    {
        i_num++;
    }
    return (i_num);
}

/*****/
/*
/* Gets the Wide-Screen-Letterbox Signalling Bit for Switching
/*
/* Input:  Pointer to storage area for Data-Bits
/*
/*
/* Returncode: WSL-Bit
/*
/*
/*****/

```

```

int      getwsl (ptr)
unsigned char *ptr;
{
    int      i,
            wsl      = 0,
            first     = 0,
            second    = 0;

    for (i = 32; i < 37; i++)      first += *(ptr + i);
    for (i = 37; i < 42; i++)      second += *(ptr + i);
    for (i = 58; i < 63; i++)      first += *(ptr + i);
    for (i = 63; i < 68; i++)      second += *(ptr + i);
    for (i = 100; i < 105; i++)     first += *(ptr + i);
    for (i = 105; i < 110; i++)     second += *(ptr + i);

    if (first < second)
    {
        wsl = 0;
    }
    else
    {
        wsl = 1;
    }

    return (wsl);
}

/*****
/*
/* Gets the Data-Groups 1 to 4
/*
/* Input:  Pointer to storage area for Data-Bits
/*
/* Output: Data-Groups 1 to 4
/*
/* Returncode: 1 on success - 0 on failure
/*
*****/

int      getdatagroups (group_ptr, all_ptr)
unsigned char *group_ptr,
            *all_ptr;
{
    int      i = 0,
            j = 0;

    for (i = 42; i < 58; i += 2)      /* Data-Group 1 */
    {
        if      ( *(all_ptr + i) == 0 ) && ( *(all_ptr + i + 1) == 1 ) )
        {
            *(group_ptr + j) = 0;      /* 01 -> 0 */
        }
        else if ( *(all_ptr + i) == 1 ) && ( *(all_ptr + i + 1) == 0 ) )
        {
            *(group_ptr + j) = 1;      /* 10 -> 1 */
        }
        else
        {
            /* 00 || 11 -> Error */
            return (0);
        }
        j++;
    }

    for (i = 68; i < 100; i += 2)      /* Data-Group 2 & 3 */
    {
        if      ( *(all_ptr + i) == 0 ) && ( *(all_ptr + i + 1) == 1 ) )
        {
            *(group_ptr + j) = 0;      /* 01 -> 0 */
        }
    }

```

```

        else if ( (*all_ptr + i) == 1) && (*all_ptr + i + 1) == 0 )
        {
            *(group_ptr + j) = 1;          /* 10 -> 1 */
        }
        else
            /* 00 || 11 -> Error */
            {
                return (0);
            }
        j++;
    }

    for (i = 110; i < 142; i += 2)          /* Data-Group 4 & CRC-8 */
    {
        if ( (*all_ptr + i) == 0) && (*all_ptr + i + 1) == 1 )
        {
            *(group_ptr + j) = 0;          /* 01 -> 0 */
        }
        else if ( (*all_ptr + i) == 1) && (*all_ptr + i + 1) == 0 )
        {
            *(group_ptr + j) = 1;          /* 10 -> 1 */
        }
        else
            /* 00 || 11 -> Error */
            {
                return (0);
            }
        j++;
    }

    return (1);
}

/*****
/*
/* Print meaning of Data-Bits to standart-output (write_device)
/*
/* Input:  Storage area for Bits of Data-Group 1 to 4
/*
/* Output: None
/*
/* Comment: Please add new allocations of Data-Bits in this routine
/*
*****/

void    databits (ptr, wsl, write_device)
unsigned char *ptr;
int wsl;
FILE *write_device;
{
    unsigned char b[32];

    memcpy (&b[0], ptr, 32);

/*****
fprintf (write_device, "\n\nLetterbox Format and Position: ");    /* b0 b1 b2 */
if (!b[ 2] && !b[ 1] && !b[ 0] && !wsl)
    fprintf (write_device, "4:3 full format / 576 lines");
else if (!b[ 2] && !b[ 1] && b[ 0] && wsl)
    fprintf (write_device, "14:9 letterbox center / 504 lines");
else if (!b[ 2] && b[ 1] && !b[ 0] && wsl)
    fprintf (write_device, "14:9 letterbox top / 504 lines");
else if (!b[ 2] && b[ 1] && b[ 0] && wsl)
    fprintf (write_device, "16:9 letterbox center / 430 lines");
else if ( b[ 2] && !b[ 1] && !b[ 0] && wsl)
    fprintf (write_device, "16:9 letterbox top / 430 lines");
else if ( b[ 2] && !b[ 1] && b[ 0] && wsl)
    fprintf (write_device, ">16:9 letterbox center");
else if ( b[ 2] && b[ 1] && b[ 0] && !wsl)
    fprintf (write_device, "16:9 full format / 576 lines anamorphic");
else

```

```

        fprintf (write_device,"not yet allocated");

/*****

        fprintf (write_device,"\nFilm Bit: ");          /* b4 */
        if ( b[ 4])
            fprintf (write_device,"25 Hz (dominancy on 1.field)");
        else
            fprintf (write_device,"50 Hz (i.e. camera)");

/*****

        fprintf (write_device,"\nSubtitling: ");        /* b5 b6 b7 */
        if      (!b[ 7] && !b[ 6] && !b[ 5])
            fprintf (write_device,"no subtitles");
        else if (!b[ 7] && !b[ 6] &&  b[ 5])
        {
            fprintf (write_device,"in active image area subtitles ");
            fprintf (write_device,"and no teletext subtitles");
        }
        else if (!b[ 7] &&  b[ 6] && !b[ 5])
        {
            fprintf (write_device,"in active image area subtitles ");
            fprintf (write_device,"and teletext subtitles");
        }
        else if (!b[ 7] &&  b[ 6] &&  b[ 5])
        {
            fprintf (write_device,"out of active image area subtitles ");
            fprintf (write_device,"and no teletext subtitles");
        }
        else if ( b[ 7] && !b[ 6] && !b[ 5])
        {
            fprintf (write_device,"out of active image area subtitles ");
            fprintf (write_device,"and teletext subtitles");
        }
        else if ( b[ 7] && !b[ 6] &&  b[ 5])
            fprintf (write_device,"only teletext subtitles");
        else
            fprintf (write_device,"not yet allocated");

/*****

        fprintf (write_device,"\nPALplus Bit: ");      /* b8 */
        if ( b[ 8])
            fprintf (write_device,"PALplus");
        else
            fprintf (write_device,"Standard Service (PAL/SECAM)");

        fprintf (write_device,"\n");

return;
}

/*****
/*
/* Detect Errors in Data-Bits (CRC-8 polynomial is: 1 + x + x4 + x5 + x7 + x8)
/*      (          = 1 1 0 0 1 1 0 1 1          )
/*      (CRC-8 as available in the 8X01A / 9401 Chip )
/*      (Signetics 8X01A / Fairchild 9401          )
/*
/*
/* Input:  Storage area for Bits of Data-Group 1 to 4 & CRC-8-Bits (32 + 8 Bits)
/*
/* Returncode: 1 on success - 0 on failure
/*
/*
/*****

int      errordetection (ptr, write_device, full_ascii)
unsigned char *ptr;
FILE      *write_device;
int       full_ascii;

```

```

{
unsigned char    d[40],
                 s[32],
                 t[ 8],
                 g[ 9] = {1,1,0,0,1,1,0,1,1};    /* CRC-8 Polynomial */
int             i, j, k,
                 crc_8 = FALSE,
                 rem_0 = TRUE;

    for (i = 0; i < 40; i++)
    {
        d[i] = *(ptr + 39 - i);
    }

    if (full_ascii)
    {
        fprintf (write_device, "\nERRORDETECTION");
        fprintf (write_device, "\nCRC-8 Polynomial: ");
        for (k = 0; k < 9; k++)
        {
            fprintf (write_device, "%d", (int) g[k]);
        }
        fprintf (write_device, "\nInput b39...b00:\n");
        for (k = 0; k < 40; k++)
        {
            if (!(k%8) && k)
            {
                fprintf (write_device, " ");
            }
            fprintf (write_device, "%d", (int) d[k]);
        }
    }

    /*****

    for (i = 0; i < 8; i++)                /* Init Shift-Register */
    {
        t[i] = d[32+i];
    }

    for (i = 0; i < 32; i++)                /* Polynomial Division */
    {
        s[31-i] = t[7];

        for (j = 7; j > 0; j--)            /* Shift */
        {
            t[j] = s[31-i] * g[j] + t[j-1];
            t[j] %= 2;
        }
        t[0] = s[31-i] * g[0] + d[31-i];
        t[0] %= 2;
    }

    /*****

    if (full_ascii)
    {
        fprintf (write_device, "\nRemainder: ");
        for (k = 0; k < 8; k++)
        {
            fprintf (write_device, "%d", (int) t[k]);
        }
        fprintf (write_device, "\n");
    }

    for (k = 0; k < 8; k++)
    {
        if (t[k] != 0)                    /* Error detected (remainder not zero) */
        {

```

```

        rem_0 = FALSE;
    }
    if (d[k] != 0)          /* CRC-8-Bits are not all zero */
    {
        crc_8 = TRUE;
    }
}

if (crc_8 && rem_0)
{
    return (1);
}
if (crc_8 && !rem_0)
{
    return (0);
}
if (!crc_8)
{
    fprintf (write_device, "\nWARNING: CRC-8 check bits are unset\n");
    return (1);
}
}
-library_code_end

-library_mods
-library_mods_end

```

A.10 helper

A.10.1 helper.pane

```

-F 4.2 1 0 170x 7+10+20 +35+1 'CANTATA Visual Programming Environment for the KHOROS System' cantata
-M 1 1 100x40+10+20 +23+1 'This is a new subform.' subform1
-P 1 0 80x38+22+2 +2+0 'Helper-Demodulator' helper
-I 1 0 0 1 0 0 50x1+2+2 +0+0 ' ' 'Input CVBS Signal' 'Input CVBS Signal' i1
-I 1 0 1 0 0 0 50x1+2+4 +0+0 ' ' 'Input Impulse Response' 'Input Impulse Response' i2
-O 1 0 0 1 0 0 50x1+2+6 +0+0 ' ' 'Output Helper Signal' 'Output Helper Signal' o
-b +2+8 'Phase-Offset of Color Carrier for Demodulation (0-360):'
-f 1 0 0 1 0 50x1+2+9 +0+0 0 360 0 'Offset' 'Offset' offset
-f 1 0 0 1 0 50x1+2+11 +0+0 0 1 1 'Clipping' 'Clipping' clipping
-f 1 0 0 1 0 50x1+2+12 +0+0 0 1 0 'Coring' 'Coring' coring
-b +2+14 'Polynomial for Non-Linear Weighting:'
-f 1 0 0 1 0 1x1+2+15 +0+0 -100 100 0 'a0' 'a0' a0
-f 1 0 0 1 0 1x1+20+15 +0+0 -100 100 1 'a1' 'a1' a1
-f 1 0 0 1 0 1x1+38+15 +0+0 -100 100 0 'a2' 'a2' a2
-f 1 0 0 1 0 1x1+2+16 +0+0 -100 100 0 'a3' 'a3' a3
-f 1 0 0 1 0 1x1+20+16 +0+0 -100 100 0 'a4' 'a4' a4
-f 1 0 0 1 0 1x1+38+16 +0+0 -100 100 0 'a5' 'a5' a5
-f 1 0 0 1 0 50x1+2+18 +0+0 1 255 127 'Amplification' 'Amplification' amplification
-T 1 0 0 1 0 40x1+2+20 +0+0 2 'Sampling Frequency (f_sc = Frequency of Color Carrier):' 'f_sc' f_sc
-l 1 0 1 1 0 1x1+2+1 +0+0 1 '4 * f_sc' 'False' 'True' '4 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+15+1 +0+0 0 '8 * f_sc' 'False' 'True' '8 * Frequency of Color Carrier' logical
-l 1 0 1 0 0 1x1+28+1 +0+0 0 '12 * f_sc' 'False' 'True' '12 * Frequency of Color Carrier' logical
-E
-H 1 10x2+39+23 'Help' 'man page for helper' $PALPLUS/doc/manpages/helper.1
-Q 1 0 10x2+20+23 'Quit'
-R 1 0 1 10x2+1+23 'Execute' 'do operation' $PALPLUS/bin/helper
-E
-E
-E

```

A.10.2 helper.prog

```
-authors
```

```

Harald Zottmann
-authors_end

-short_prog_description
Demodulates the helper signal from the CVBS signal.
-short_prog_description_end

-short_lib_description
Demodulates the helper signal from the CVBS signal.
-short_lib_description_end

-man1_long_description
.I helper
will demodulate the helper signal (72 lines / field;
144 lines / frame) from a CVBS signal provided by raw2cvbs.
.LP
The programm detects the phase of the color subcarrier and
allows to specify an offset (0-360 degrees) to this phase for
demodulation. The helper signal will be filtered with the input
impulse response filter and scaled to -1...+1 (-150mv...+150mv).
Coring sets all signal values below the coring value to zero.
The parameters a0-a5 describe a polynomial which is an approximation
for the desired non-linear weighting. The polynomial should be
defined between 0 and 1. Negative values will be handled symmetrical.
After the non-linear weighting, clipping sets all signal values above
the clipping value to the value of clipping. Finally the signal will
be amplified.
.LP
The output helper signal will be of the same length as the input cvbs
signal. No selection between helper rows and other rows is performed.
.LP
(Herfet,Th.: Breitbild-PAL mit vertikaler Bandaufspaltung.
FKT, Vol. 46 (1992), No.10, pp 673-679)
-man1_long_description_end

-man1_examples
-man1_examples_end

-man1_restrictions
.I helper
.LP
Datatypes of input and output signals:
.LP
input cvbs signal:      1_BYTE (0...255)
.LP
input impulse response: FLOAT
.LP
output helper signal:   1_BYTE (0...255)
                        (128-127...128+127)
-man1_restrictions_end

-man1_see_also
-man1_see_also_end

-man3_long_description
.I helper
will demodulate the helper signal (72 lines / field;
144 lines / frame) from a CVBS signal provided by raw2cvbs.
.LP
The programm detects the phase of the color subcarrier and
allows to specify an offset (0-360 degrees) to this phase for
demodulation. The helper signal will be filtered with the input
impulse response filter and scaled to -1...+1 (-150mv...+150mv).
Coring sets all signal values below the coring value to zero.
The parameters a0-a5 describe a polynomial which is an approximation
for the desired non-linear weighting. The polynomial should be
defined between 0 and 1. Negative values will be handled symmetrical.
After the non-linear weighting, clipping sets all signal values above
the clipping value to the value of clipping. Finally the signal will
be amplified.

```

```

.LP
The output helper signal will be of the same length as the input cvbs
signal. No selection between helper rows and other rows is performed.
-man3_long_description_end

-man3_restrictions
-man3_restrictions_end

-man3_see_also
-man3_see_also_end

-usage_additions
-usage_additions_end

-include_includes
-include_includes_end

-include_additions
-include_additions_end

-include_macros
/*****
/* Read Input-Images */
*****/

#define READINPUT(cvbs_img); \
    cvbs_img = readimage(helper->i1_file);\
    if(cvbs_img == NULL)\
    {\
        (void) fprintf(stderr,"helper : cannot read input image");\
        (void) fprintf(stderr," %s\n",helper->i1_file);\
        exit(1);\
    }

#define READINPUT_FIRFIL(imp_res_img); \
    imp_res_img = readimage(helper->i2_file);\
    if(imp_res_img == NULL)\
    {\
        (void) fprintf(stderr,"helper : cannot read input image");\
        (void) fprintf(stderr," %s\n",helper->i2_file);\
        exit(1);\
    }

/*****
/* Check Input-Images */
*****/

#define CHECKINPUT(cvbs_img); \
    (void) proptype(program, cvbs_img, VFF_TYP_1_BYTE, TRUE);\
    (void) proper_loc_type(program, cvbs_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, cvbs_img, 1, TRUE);\
    (void) proper_map_enable(program, cvbs_img, VFF_MAP_OPTIONAL, TRUE);

#define CHECKINPUT_FIRFIL(imp_res_img); \
    (void) proptype(program, imp_res_img, VFF_TYP_FLOAT, TRUE);\
    (void) proper_loc_type(program, imp_res_img, VFF_LOC_IMPLICIT, TRUE);\
    (void) proper_num_bands(program, imp_res_img, 1, TRUE);\
    (void) proper_map_enable(program, imp_res_img, VFF_MAP_OPTIONAL, TRUE);
-include_macros_end

-main_variable_list
struct xvimage *cvbs_img,
               *imp_res_img,
               *helper_img,
               *readimage();
-main_variable_list_end

-main_before_lib_call
    if(check_args()) exit(1);

```

```

READINPUT (cvbs_img);
CHECKINPUT (cvbs_img);

if (helper->i2_flag == 1)
{
    READINPUT_FIRFIL (imp_res_img);
    CHECKINPUT_FIRFIL (imp_res_img);
}
-main_before_lib_call_end

-main_library_call
if(! lhelper(cvbs_img, imp_res_img,
             &helper_img,
             helper->i2_flag,
             helper->offset_float,
             helper->clipping_float,
             helper->coring_float,
             helper->a0_float, helper->a1_float,
             helper->a2_float, helper->a3_float,
             helper->a4_float, helper->a5_float,
             helper->amplification_float,
             helper->f_sc_toggle) )
{
    fprintf(stderr, "lhelper failed.\n");
    exit(1);
}
-main_library_call_end

-main_after_lib_call
if(! writeimage(helper->o_file, helper_img))
{
    (void) fprintf(stderr, "helper : unable to write helper output file\n");
    exit(1);
}
-main_after_lib_call_end

-library_includes
-library_includes_end

-library_input
-library_input_end

-library_output
-library_output_end

-library_def
int lhelper(    cvbs_img,
                imp_res_img,
                helper_img,
                lowpass_filter,
                phase_offset,
                clipping,
                coring,
                a0, a1, a2,
                a3, a4, a5,
                amplification,
                f_sc_toggle)

struct  xvimage *cvbs_img,
        *imp_res_img,
        **helper_img;

int      lowpass_filter,
        f_sc_toggle;

float    phase_offset,
        clipping,
        coring,
        a0, a1, a2,
        a3, a4, a5,

```

```

        amplification;
-library_def_end

-library_code
/*****
/*
/* Library Code for helper
/*
/*
*****/

#include "vinclude.h"

/*****

#define FCARRIER      4433618.75
#define FROW           15625

#define FC_FAC_4        1
#define FC_FAC_8        2
#define FC_FAC_12       3

#define NO_BURST        100.0

/*****
/*
/* Main Function
/*
/*
*****/
{
double      getburstphase  ();
float       getblacklevel  ();
int         convo          ();

float       signal         = 0.0,
           carrier         = 0.0,
           blacklevel      = 0.0,
           *fil_ptr        = NULL;
double      phase         = 0.0,
           omega           = 0.0,
           carrier_fac     = 0.0,
           samp_freq       = 0.0;
unsigned char *cvbs_ptr    = NULL,
           *helper_ptr    = NULL;
unsigned long il           = 0L,
           fil_length     = 0L,
           cvbs_length    = 0L;
int         neg           = FALSE;

/*****

fprintf(stdout, "\nHELPER V1.3:\n");

switch (f_sc_toggle)
{
    case FC_FAC_4:
        carrier_fac = 4.0;
        break;

    case FC_FAC_8:
        carrier_fac = 8.0;
        break;

    case FC_FAC_12:
        carrier_fac = 12.0;
        break;

    default:
        fprintf (stderr, "Missing or improper argument in call to helper.\n");
        return (0);
}

```

```

    samp_freq = FCARRIER * carrier_fac;

/*****

    if (lowpass_filter)
    {
        fil_length = imp_res_img->row_size;
        if (!(fil_length % 2L))
        {
            fprintf (stderr, "Filter-Length must be odd-numbered.\n");
            return (0);
        }
        fil_ptr = (float *) imp_res_img->imagedata;
    }

/*****

    cvbs_length = (unsigned long) (cvbs_img->col_size * cvbs_img->row_size);
    cvbs_ptr     = (unsigned char *) cvbs_img->imagedata;

    blacklevel  = getblacklevel (cvbs_ptr, carrier_fac);

/*****

    *helper_img = createimage ((unsigned long) (cvbs_length), 1L, VFF_TYP_1_BYTE,
                               1,1, "helper_signal", 0, 0, VFF_MS_NONE, VFF_MAPTYP_NONE,
                               VFF_LOC_IMPLICIT, 0);

    if (*helper_img == NULL)
    {
        fprintf (stderr, "Out of memory.\n");
        return(0);
    }

    helper_ptr = (unsigned char *) (*helper_img)->imagedata;

/*****
/* Get the Phase of the Color-Carrier (phase) */
/*****

    phase  = getburstphase (cvbs_ptr, carrier_fac, (unsigned char) blacklevel, 5);
    if (phase == NO_BURST)
    {
        phase = getburstphase (cvbs_ptr, carrier_fac, (unsigned char) blacklevel, 6);
        if (phase == NO_BURST)
        {
            fprintf (stderr, "No burst found in CVBS-Signal.\n");
            return (0);
        }
        fprintf (stdout, "Starting with field 1\n");
        phase -= 3.0 * M_PI / 4.0;          /* -135 DEG */
    }
    else
    {
        fprintf (stdout, "Starting with field 3\n");
        phase -= 3.0 * M_PI / 4.0;          /* -135 DEG */
    }

    omega = (double) (2.0 * M_PI / carrier_fac);
    phase += (double) (phase_offset * M_PI / 180.0);

/*****
/* Demodulate Helper-Signal */
/*****

    for (il = 0L; il < cvbs_length; il++)
    {
        signal = (float) *(cvbs_ptr + il);
        signal -= blacklevel;
        signal /= 38.25;                    /* Ux/Uo -> -1...+1 */
    }

```

```

    carrier = (double) sin (omega * ((double) il) + phase);

    signal += carrier;                                /* Demodulation */

    signal *= 127.0;
    signal += 128.0;

    if (signal > 255.0)    signal = 255.0;    /* Check for overflow */
    if (signal < 0.0)    signal = 0.0;    /* Check for undeflow */

    *(helper_ptr + il) = (unsigned char) signal;    /* Casting */
}

/*****
/* Convolution with Impulse-Response
*****/

if (lowpass_filter)
{
    if (!convo (helper_ptr, fil_ptr, cvbs_length, fil_length))
    {
        fprintf (stderr, "Convolution (Helper-Signal) failed.\n");
        return (0);
    }
}

/*****
/* Coring / Non-Linear-Weighting / Clipping
*****/

for (il = 0L; il < cvbs_length; il++)
{
    signal = (float) *(helper_ptr + il);
    signal -= 127.0;
    signal /= 64.0;                                /* Scaling after demodulation */

    if (signal < 0.0)
    {
        signal *= -1.0;
        neg = TRUE;
    }

    if (signal < coring)                                /* Coring */
    {
        signal = 0.00;
    }

    else                                /* Non-Linear-Weighting */
    {
        signal = a0
            + a1 * signal
            + a2 * (float) pow ((double) signal, 2.0) +
            + a3 * (float) pow ((double) signal, 3.0) +
            + a4 * (float) pow ((double) signal, 4.0) +
            + a5 * (float) pow ((double) signal, 5.0);
    }

    if (signal > clipping)                                /* Clipping */
    {
        signal = clipping;
    }

    if (neg)
    {
        signal *= -1.0;
        neg = FALSE;
    }

    signal *= amplification;                                /* Amplification */
    signal += 128.0;
}

```

```

        if (signal > 255.0)    signal = 255.0;    /* Check for overflow */
        if (signal < 0.0)    signal = 0.0;    /* Check for undeflow */

        *(helper_ptr + il) = (unsigned char) signal;    /* Casting */
    }

    fprintf(stdout, "Done: HELPER\n");

return (1);
}

/*****
/*
/* Performs a circular convolution of 1_BYTE-Signals (unsigned char)
/* with an Impulse-Response (float).
/*
/*
/* Input:  Data vector of Color-Component      (datain1)
/*         Data vector of Impulse-Response    (datain2)
/*         Number of data points              (dpoints1, dpoints2)
/*
/*
/* Output: Resultant convolution data          (datain1)
/*
*****/

int    convo (datain1, datain2, dpoints1, dpoints2)
signed long    dpoints1,
              dpoints2;
unsigned char  *datain1;
float          *datain2;
{
    signed long    shift,
                  idx,
                  n, h;
    float          *conv,
                  temp;

    if (dpoints1 < dpoints2)
    {
        fprintf(stderr, "Data vector 1 must be longer than Data vector 2\n");
        return (0);
    }

    conv = (float *) malloc((unsigned)(dpoints1 * sizeof(float)));
    if (conv == NULL)
    {
        fprintf(stderr, "Out of memory.\n");
        return (0);
    }

    for (n = 0L; n < dpoints1; n++)
    {
        temp = 0.0;
        for (h = 0L; h < dpoints2; h++)
        {
            idx = n - h;
            if (idx < 0L)
            {
                idx += dpoints1;
            }
            temp += datain2[h] * (float) datain1[idx];
        }
        conv[n] = temp;
    }

    for (n = 0L; n < dpoints1; n++)
    {
        temp = conv[n];
        if (temp > 255.0)
        {

```

```

        temp = 255.0;
    }
    if (temp < 0.0)
    {
        temp = 0.0;
    }
    datain1[n] = (unsigned char) temp;
}

shift = dpoints2 / 2;
for (n = 0L; n < dpoints1 - shift; n++)
{
    datain1[n] = datain1[n + shift];
}

free (conv);

return (1);
}

/*****
/*
/* Gets the phase of the Color-Carrier (burst)
/*
/* Input:  Pointer (ptr) to CVBS-Signal
/*         Samplefrequency (fsc_fac * Frequency of Color-Carrier)
/*         Blacklevel (black_level)
/*         Number of the row to search burst for (row_number)
/*
/* Output: phase of the Color-Carrier at the beginning of the CVBS-Signal
/*
/*
*****/

double getburstphase (ptr, fsc_fac, black_level, row_number)
unsigned char *ptr;
double fsc_fac;
unsigned char black_level;
signed int row_number;
{
double phase = 0.0,
ref_omega = 0.0,
fsc_omega = 0.0,
burst_amp = 0.0,
err_val = 0.0,
err_min = 9.9E99,
a1, a2,
pixel_row = 0.0,
ref_sig [768],
fsc_sig [ 13];
signed long burst_offset = 0L,
num_sin = 0L,
i, j,
step = 0L,
pos = 0L;

num_sin = (long) (4 * 8 * 12);
fsc_omega = (double) (2.0 * M_PI / fsc_fac);
ref_omega = (double) (2.0 * M_PI / (double) num_sin);
pixel_row = (double) FCARRIER / FROW * fsc_fac;
burst_amp = (double) black_level / 2.0;
burst_offset = (long) (row_number * pixel_row) + (long) (25 * fsc_fac);
step = (long) (num_sin / (long) fsc_fac);

memset (&ref_sig[0], 0x00, 2 * num_sin * sizeof(double));

for (i = 0; i < 2 * num_sin; i++) /* Reference-Sinus-Signal */
{
    ref_sig[i] = burst_amp * sin(ref_omega * (double) i);
}

```

```

memset (&fsc_sig[0], 0x00, 13 * sizeof(double));
for (i = 0; i <= (long) fsc_fac; i++)          /* One period of burst */
{
    fsc_sig[i] = (double) *(ptr+burst_offset+i) - (double) black_level;
}

for (j = 0; j < num_sin; j++)
{
    err_val = 0.0;
    for (i = 0; i <= (long) fsc_fac; i++)
    {
        a1 = fsc_sig[i];
        a2 = ref_sig[j + i * step];
        err_val += fabs(a1 - a2);
    }
    if (err_val < err_min)
    {
        err_min = err_val;
        pos = j;
    }
}

if (err_min > 20.0 )
{
    return (NO_BURST);
}

phase = ref_omega * (double) pos;
phase -= fmod(fsc_omega * (double) (burst_offset), 2.0 * M_PI);

return (phase);
}

/*****
/*
/* Gets the level of BLACK (row number 6)
/*
/* Input:  Pointer (ptr) to CVBS signal
/*         Samplefrequency (fsc_fac * Frequency of Color-Carrier)
/*
/* Returncode: level of BLACK
/*
/*
*****/

float  getblacklevel (ptr, fsc_fac)
unsigned char  *ptr;
double        fsc_fac;
{
    signed long    mean_start    = 0L,
                  mean_end      = 0L,
                  il;
    float          pixel_row     = 0.0,
                  black_level    = 0.0;

    pixel_row      = (float)      (FCARRIER / FROW * fsc_fac);
    mean_start     = (signed long) ( 5.0 * pixel_row) + (long) ( 50 * fsc_fac);
    mean_end       = (signed long) ( 5.0 * pixel_row) + (long) (250 * fsc_fac);

    for (il = mean_start; il < mean_end; il++)
    {
        black_level += (float) *(ptr + il);
    }
    black_level /= (float) (mean_end - mean_start);

    return (black_level);
}
-library_code_end

-library_mods
-library_mods_end

```