Digital Processing of Analog Television

Lior Zimet
EE392J Project Report, Winter 2002
Department of Electrical Engineering, Stanford University, CA 94305, USA

Abstract

NTSC composite color video is currently one of the most common formats of video broadcasting. Although the signal is analog, it can be decoded almost entirely digitally. Analog decoders offer limited video quality and require adjustments. Using digital techniques to implement NTSC decoding offers minimum adjustments and excellent video quality. This project presents the building blocks of digital processing of an NTSC color signal. The processing stages include sampling the analog signal, synchronization information detection, luminance and chrominance separation, color demodulation, and interlace to progressive conversion for display on PC monitor in RGB color space.

1. Introduction

Analog decoders of color video signal have been around for some time. However, they have been difficult to use and provide limited video quality. Digital video does far more compared with analog. Its most exciting aspects are the tremendous possibilities, which are denied to analog technology. By sampling the 1D analog composite signal, we can perform the decoding digitally on a pixel base, and apply many different algorithms to get better video quality.
The NTCS composite video standard, defined in 1952, is 2:1 interlaced with 262.5 lines per field (525 lines per frame), 60 fields per second, and 4:3 aspect ratio. The DC level of the analog signal defines the monochrome luminance information. The Hue and saturation information of the color is transmitted using a 3.58MHz sub-carrier within the same bandwidth as the monochrome signal. I and Q (or U and V) are used to modulate the color sub-carrier in phase quadrature. Timing information such as vertical sync and horizontal sync are inserted every field or line period respectively to define the beginning of a field, beginning of a line, and blanking information.

$$NTSC = Y + U\sin(\omega t) + V\cos(\omega t) + TIMING$$

To display a composite NTSC signal on a progressive RGB display such as a computer monitor, several stages of processing need to be carried out. Figure 1 depicts a block diagram of digital processing system for an analog TV signal. The first 2 stages of preparing the analog signal and converting it to digital are done by hardware and will be discussed in section 2. The samples out of the ADC are digital quantized form of the 1D signal. To extract the valid video data, in stage 3, the timing information is used and the data is organized in a field matrices. Each raw in the matrix represents the active video of one horizontal line. This stage is discussed in section 3. For easier processing of the chrominance and luminance separation, the data is resampled to four times the sub-carrier frequency. Y/C separation is the most challenging part of composite video processing. In section 5 we discuss the different methods for implementing Y/C separation. In the next stage the color information is being demodulated from the sub-carrier as discussed in section 7. To display the information on a progressive display such as a PC monitor, interlace to progressive conversion as well as YUV to RGB conversions are performed. These two stages are discussed in sections 8 and 9.

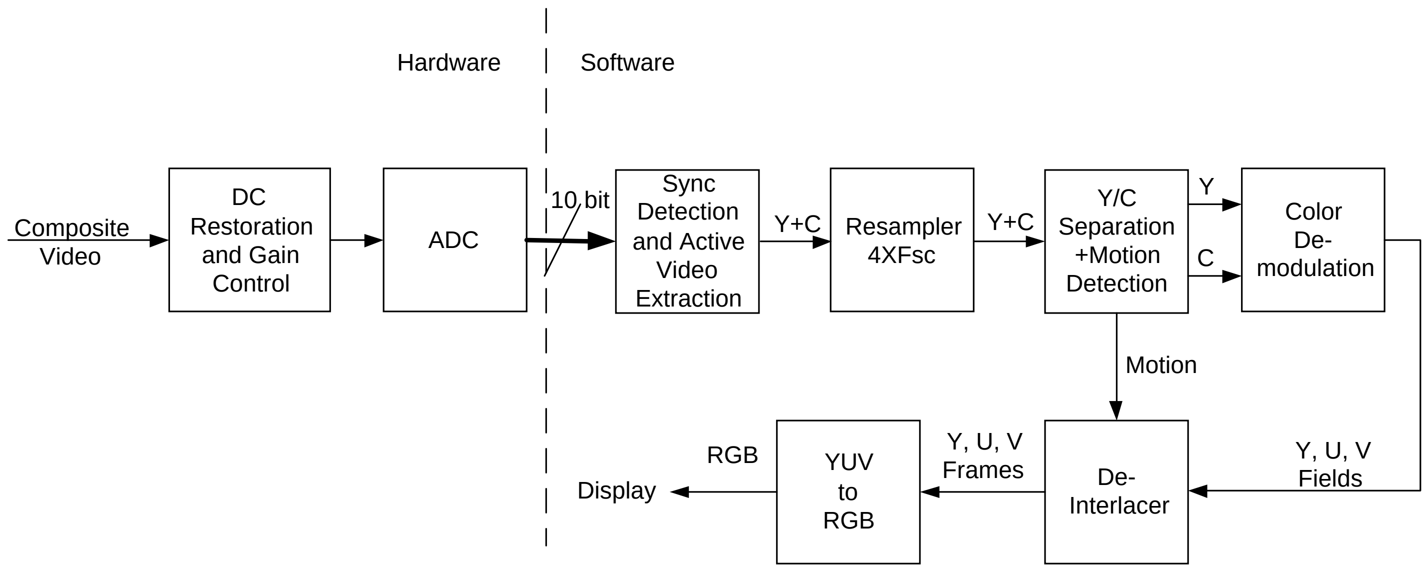


Figure 1 - System block diagram

2. Sampling the analog composite video

The Analog front end of the system, which is implemented in hardware, performs the following functions:

- DC restore
- Gain control
- Analog to digital conversion

The DC restoration is required because the video signal is usually AC coupled. The problem with this is that the video is inherently a DC signal. For proper decoding, certain reference parts of the signal, such as the horizontal sync, must always be at the same voltage. The DC restore circuit is used to place these pats of the video signal at a voltage level so that the ADC will create the expected code.

The gain control is needed because the baseband video signal may come in different levels, depends on the video source and system terminations. The gain may be manually controlled per source, or in more complex systems it can be adjusted in real time with a feedback from the digital processing block. The overall gain of the signal should be such that it takes advantage of the maximum dynamic range of the ADC. In some systems, the gain control is done in the digital domain after the ADC.

For video applications a 10bit ADC is usually sufficient. The smallest value represents the sync level. The luminance signal occupies the amplitude from blank level to peak white, and the portion above the blank level is reserved for the chrominance as depicted in figure 3. An analog operational amplifiers circuit usually realizes both the DC restore and the gain control. The desired sampling clock feeds the ADC, and the digital samples are then captured for digital processing.
In this project, a front end of an existing system with a 27MHz-sampling rate was used, and the digital data was captured to a file to perform the digital processing using Matlab.

3. Sync detection and Active video extraction

The sampled analog composite video is captured to a binary file. Every word (2 bytes) in the file represents one sample of 10 bits (6 most significant bits are ignored). The samples are 37ns spaced in time (27MHz sampling clock). Using the timing information embedded in the composite signal, the active video of each line can be extracted and placed in a matrix for further processing.
Figure 2 depicts the vertical blanking information as plotted directly from the sampled video. The blanking period starts from line 1 of each field and includes the vertical sync (vsync) pulse, which defines the beginning of a new field. Lines 10-21 are usually used for VBI data (vertical blanking interval) such as close caption and teletex, and line 22 is the first line with active video. The active video lines continue until line 262.5 where the next field blanking information starts. This interval ends in line 525.
The bottom of the sync is the smallest value in the sampled data stream and can be identified by using a threshold. The vsync pulse can be detected by looking for the specific timing information of six wide negative pulses with positive pulses (hsync wide) between them.

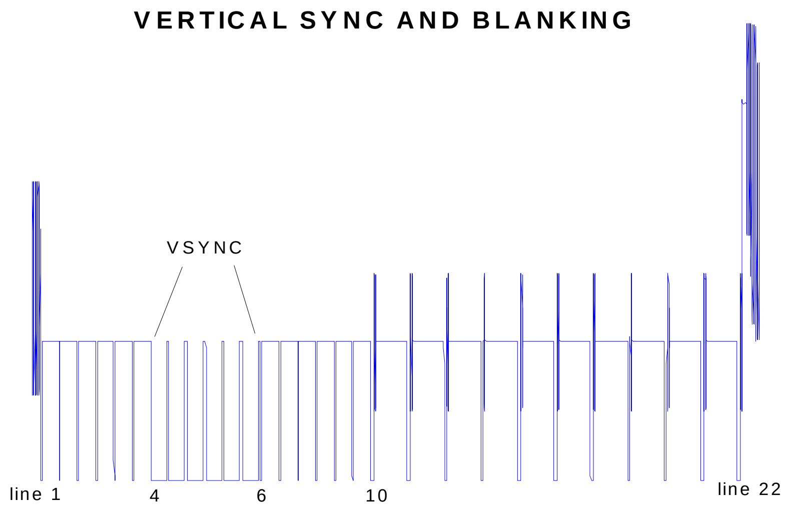


Figure 2 - Vertical sync and blanking

Figure 3 depicts one video line as plotted directly from the sampled data. The line starts from the horizontal sync (hsync), continues with the color burst information, which follows by the active video information. The hsync detction is done again by using a threshold. By knowing the width of the pulse (4.7us) and the relative active video starting point, we can extract the active video samples from each line and arrange it in a matrix format. The final matrix for each video sequence is a 3D matrix such that each 2D matrix is one field and each raw is one active line. For 27MHz sampling rate, each field occupies 1440 pixels per line of active video and 254 lines (excluding the blanking lines).

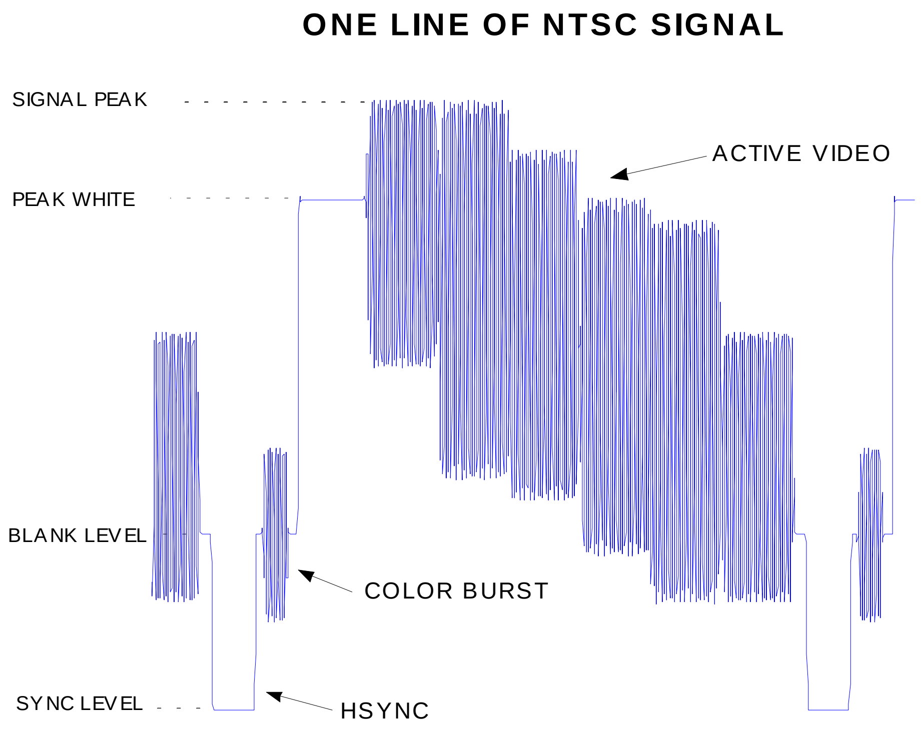


Figure 3 - One line of NTSC signal

4. Digital Phase Lock Loop

As depicted in figure 3, each video line contains color burst information. The color burst information is located in the “back porch” of the video line, which starts after the hsync and ends before the active video. As depicted in figure 4, the color burst consists of 9 cycles of the sub-carrier with the specific phase that was used to modulate the color in the current video line. The decoder can extract the color burst phase and frequency of each line, and perform an accurate demodulation of the color. In NTSC, the phase of the color burst is inverted in every line, and at the beginning of every odd field. Four complete fields are required to repeat the interval of the color burst phase. We will see later that the alternating phase of the color burst will be very useful for separating luminance from chrominance.

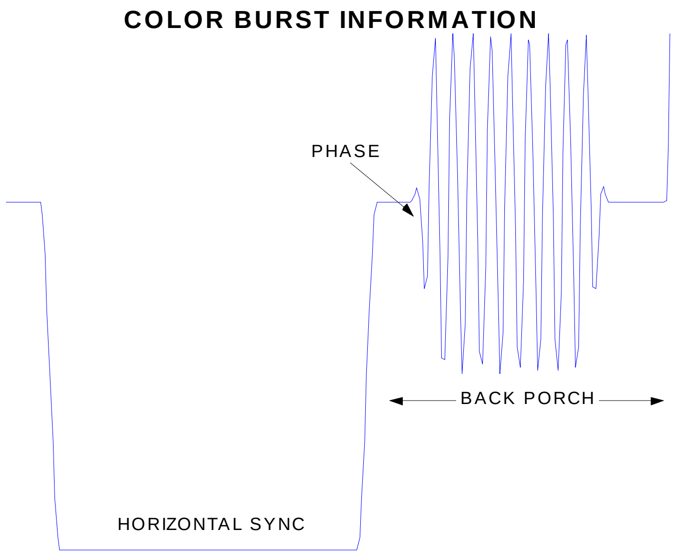


Figure 4 - Color burst information

To extract the color burst phase and frequency for each line, a digital PLL (Phase Lock Loop) must be implemented. For each video line, the phase in which the color bust starts (0 or 180) and the exact frequency are stored. The phase can be easily detected by looking at the first peak of the color burst. To extract the frequency in an accurate way, the color burst is interpolated to get a very smooth sine wave, and zero crossing around the blank level is detected. The desired frequency can be found by calculating the mean crossing distance between every cycle in the interpolated color burst.
Experimental results of color demodulation with the extracted color burst frequency show moderate performance of the color accuracy, and a better algorithm for frequency tracking can be developed.

5. Y/C separation

The luminance and the chrominance information are typically combined in the encoder by simply adding them together to form the composite signal. The result is that chrominance and high frequency luminance signals occupy the same portion of the frequency spectrum. As a result, separating them in the decoder is difficult.
There are two commonly seen artifacts when displaying video picture from the composite video source namely, cross-luminance and cross-chrominance. Cross-luminance happens when the modulated color signal is not completely removed from the composite signal and being decoded as luminance. Cross-chrominance happens when high-frequency luminance signal from the chrominance portion of the frequency band is not removed completely, and decoded as chrominance.
A better video quality can be achieved by performing the Y/C separation digitally, using more complex techniques.

The NTSC frequency spectrum occupies 6MHz bandwidth. The luminance and chrominance information is interleaved in the spectrum as depicted in figure 5. That will allow the usage of use better techniques for Y/C separation as discussed in sections 5.3-5.5.

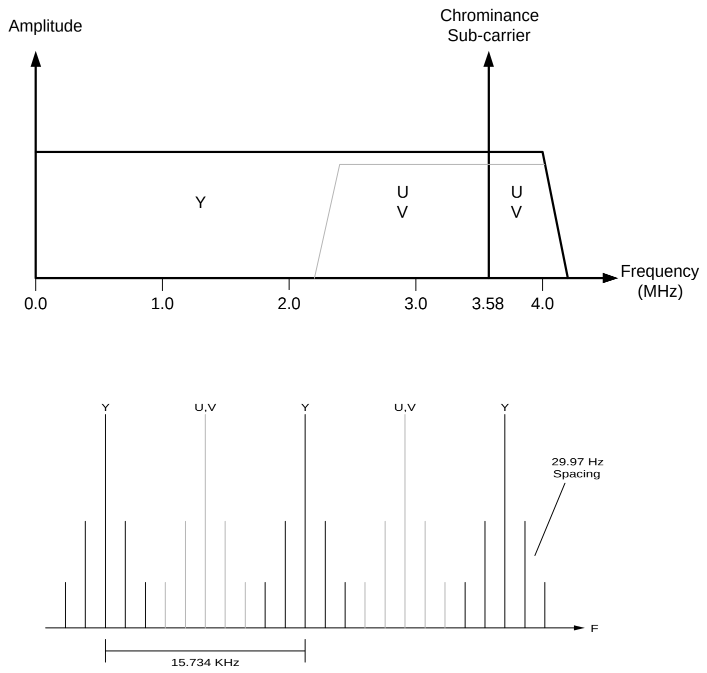


Figure 5 - NTSC signal spectrum

5.1. Re-sampler

The NTSC composite video signals is formed by superimposing the quadrature modulated chrominance signal onto the luminance signal. The U and V are modulated using different phase of the sub-carrier. For simpler Y/C separation and color demodulation, the input analog composite signal is resampled from 27MHz to four times the chrominance sub-carrier frequency. In section 5.3-5.5 we will see that the resampling structures every pixel as $Y \pm U$ or $Y \pm V$, which makes the separation easier. In section 7 we will see that the sine and cosine coefficients for the color demodulation can be as simple as [0, 1, -1].
To perform the down sampling from 27MHz to 4x3.58MHz, an up-sampling by 35 and a down-sampling by 66 are performed.

5.2. Bandpass and Notch filters

The conventional way to perform Y/C separation is using a notch filter to extract the luminance, and a bandpass filter to extract the chrominance. Usually, the notch filter and the bandpass filter are centered at the chrominance sub-carrier with a span of $\pm 1.3MHz$.
For better performance, a non-symmetric filter can be used to match the color spectrum as depicted in figure 5.
Obviously, when using notch filter, some of the high frequency luminance information will be lost, and cross color and cross luminance will appear in the decoded video.

Figure 6 shows an example of the luminance information of a sine test pattern with the use of a notch filter compared with a 3D comb filter (discussed in section 5.5).

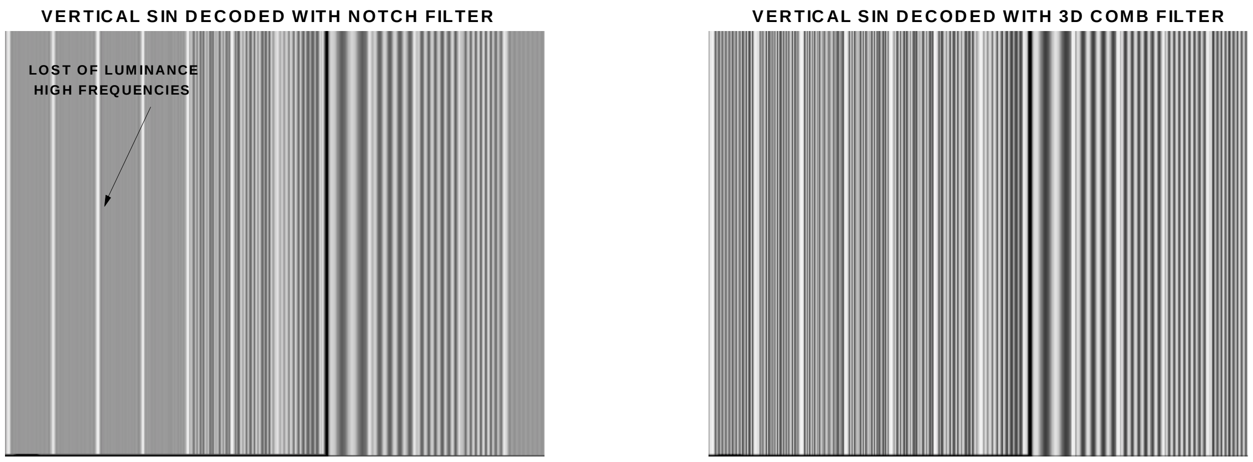

VERTICAL SIN DECODED WITH NOTCH FILTER
LOST OF LUMINANCE HIGH FREQUENCIES
VERTICAL SIN DECODED WITH 3D COMB FILTER

Figure 6 - Sine pattern decode with notch and 3D comb filters

5.3. 1D comb filter

To take advantage of the way the chrominance and the luminance were combined in the composite signal spectrum, a comb filter can be used. Looking at the pixel domain, the luminance and chrominance information are combined in such a way that spatial processing can be used to perform the separation. Figure 7 depicts pixels of three consecutive lines.

$$\begin{tikzcd}
Y+U \arrow[d, leftrightarrow] & Y-V \arrow[d, leftrightarrow] & Y-U \arrow[d, leftrightarrow] & Y+V \arrow[d, leftrightarrow] \\
Y-U \arrow[d, leftrightarrow] & Y+V \arrow[d, leftrightarrow] & Y+U \arrow[d, leftrightarrow] & Y-V \arrow[d, leftrightarrow] \\
Y+U & Y-V & Y-U & Y+V
\end{tikzcd}$$

Figure 7 - Vertical comb filter using 3 lines

As seen, the chrominance signals U, V can be separated from the luminance signal Y by adding the center line pixel to either the upper line pixel or the lower line pixel. This operation assumes that there is no change in Y or U, V between adjacent pixels.

$$Y =((Y_{center}+C_{center}) + (Y_{upper}-C_{upper})) / 2$$

$$C = ((Y_{center}+C_{center}) - (Y_{upper}-C_{upper})) / 2$$

Or

$$Y =((Y_{center}+C_{center}) + (Y_{lower}-C_{lower})) / 2$$

$$C = ((Y_{center}+C_{center}) - (Y_{lower}-C_{lower})) / 2$$

Or for better results, a three-line comb filter is implement by using the following equations:

$$Y = ((Y_{center}+C_{center}) + (Y_{upper}-C_{upper}) + (Y_{center}+C_{center}) + (Y_{lower}-C_{lower})) / 4$$

$$C = ((Y_{center}+C_{center}) - (Y_{upper}-C_{upper}) + (Y_{center}+C_{center}) - (Y_{lower}-C_{lower})) / 4$$

5.4. 2D adaptive comb filter

Although the simple vertical comb filter provides a better degree of luminance and chrominance separation compared to traditional notch and bandpass filter approach, it performs perfectly only for vertically orientated structures in a picture. If the information in adjacent lines is different, decoding errors known as hanging dots occur. Therefore, adaptive technique must be employed to improve the picture quality. The two-dimensional adaptive filter combines both the vertical comb filter with adaptive weighting, and the horizontal notch filter.

The simple 3-line comb filter is based on equally weighing result from the upper line and lower line pixel relative to the center line pixel. Therefore, if the color transition between the lines is large, the result may be incorrect. By employing adaptive method, the weights for mixing two pixels can be determined according to the amount of color change on the upper or lower pixels. To determine the amount of color change, the chrominance information from the bandpass filter of the pixels is being compared and a weight coefficient is determined. For example, if the center pixel of the line is the same as the upper line pixel but different than the lower line pixel, the adaptive comb will generate a weight of zero for the lower line pixel and a weight of one for the upper line pixel. With the simple comb filter, it will simply average the upper and lower pixels and hanging dots artifact will be seen. The above method can be summarized with the following equations:

$$Chroma = aC_1 + bC_2$$

a, b = the weighting coefficients
C1, C2 = the chrominance obtained by using simple 2-line comb filter

Whenever the color transition indicates that horizontal edge is detected, and both C1 and C2 will not perform a good Y/C separation, the algorithm use the simple notch filter for Y/C separation.

5.5. 3D adaptive comb filter

3D adaptive Y/C separation further improves the luminance-chrominance separation process such that the separation is done perfectly for the stationary parts of the image.

If the picture is stationary, the pixels between frames (temporal direction) are identical, except that the modulated chrominance phase is inverted (180 degrees). Figure 8 depicts the luminance and chrominance information in the pixel domain with regards to time.

By adding and subtracting pixels between frames, the luminance and chrominance can be perfectly separated from the composite signal.

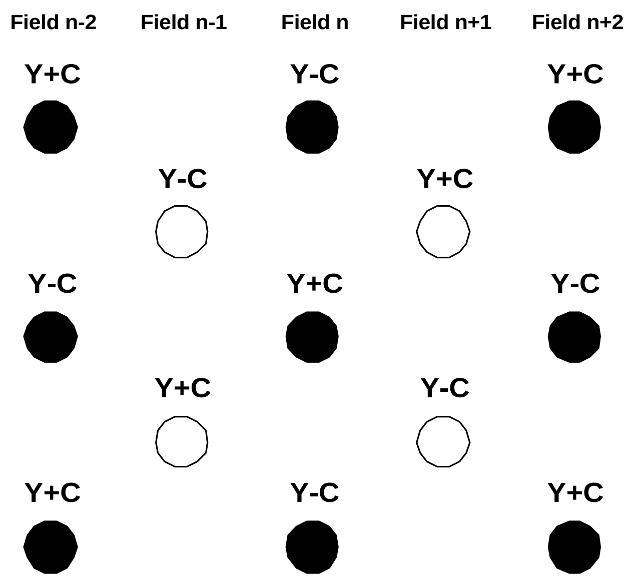


Figure 8 - Luminance and Chrominance information on the temporal axis

However, if the area of the picture contains motion, pixels between frames are no longer identical, and a 2D adaptive Y/C separation will provide better results. Therefore, a motion detector is required to identify the areas of the picture containing motion, such that the Y/C separator can adaptively mix between temporal and spatial filters.
Figure 9 depicts a simple block diagram of the 3D Y/C separator.

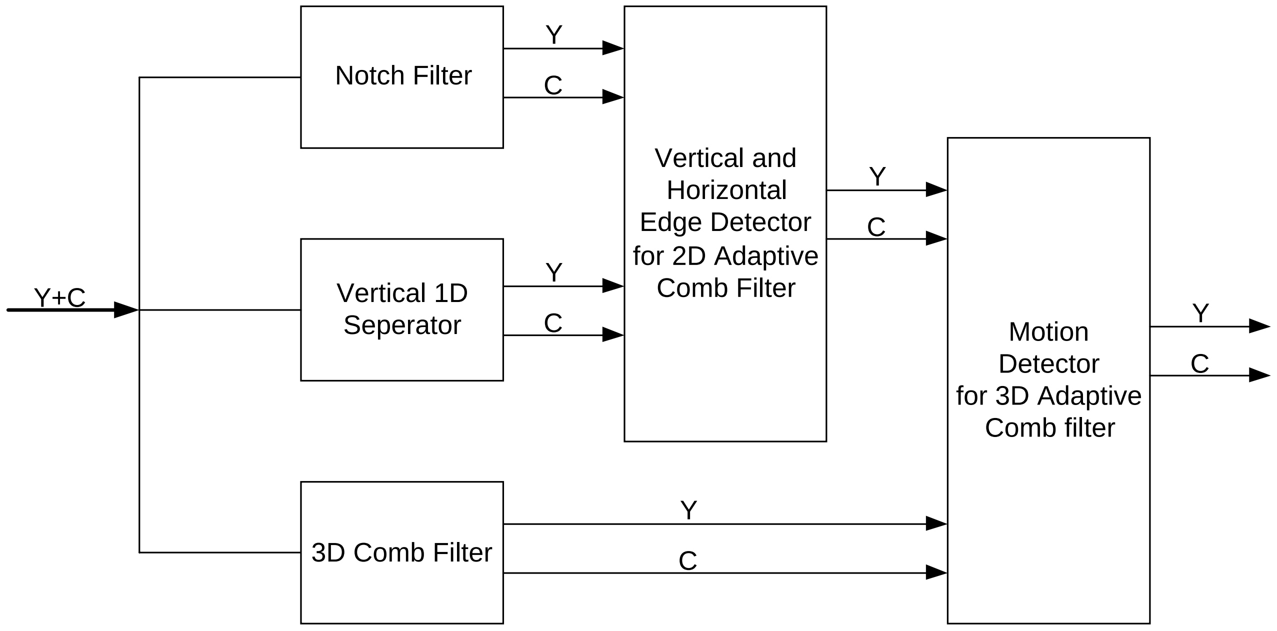

Y+C
Notch Filter
Y
C
Vertical 1D Seperator
Y
C
3D Comb Filter
Y
C
Vertical and Horizontal Edge Detector for 2D Adaptive Comb Filter
Y
C
Motion Detector for 3D Adaptive Comb filter
Y
C

Figure 9 - Y/C separation block diagram

Figure 10 shows the “Cheer leader” sequence decoded with a 3D comb filter without the use of motion detection. The motion artifacts can be easily seen where motion appear in the scene.

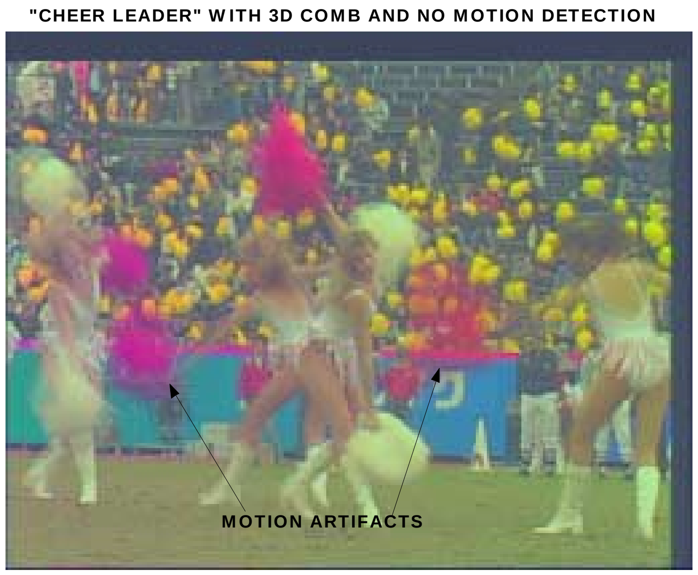
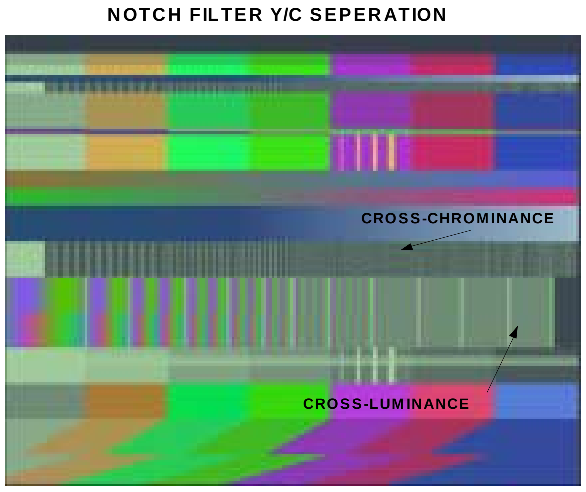
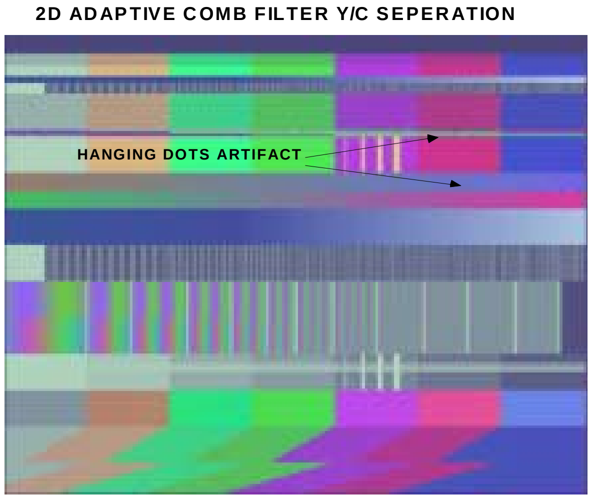


Figure 10 - Motion artifacts in 3D comb filtering

Figure 11 shows a still image test pattern that was decoded using 3 different methods: Notch filter, 2D comb filter and 3D adaptive comb filter. Using the Notch filter, we can see that the image is blurred, high luminance frequency is missing and the artifacts of cross-luminance and cross-chrominance. With 2D comb filter, in difficult areas such as the single line pattern (towards the top of the image), hanging dots appear. The 3D adaptive separation is giving a perfect result on this still image pattern.





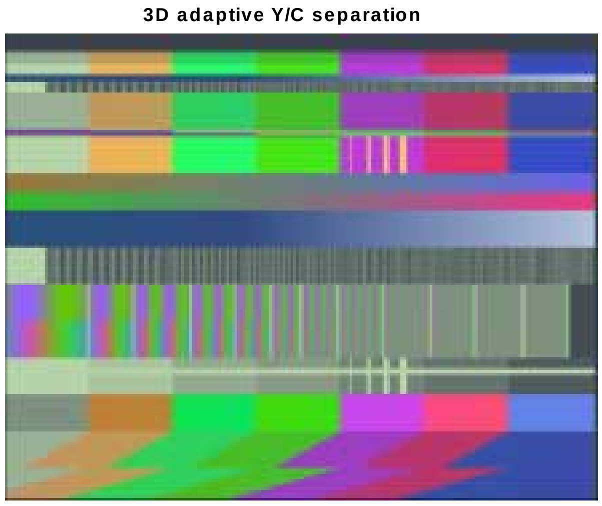


Figure 11 - Test pattern decoded with 3 different Y/C separation methods

6. Motion detector

One of the key components in 3D Y/C separation and de-interlacing is the motion detector. The motion detector that was implemented is based on the *sum of absolute difference* method. For each pixel, large motion and small motion are detected. Large motion detection is depicted in figure 12. For each pixel in field n, the value of the same pixel from field n-2 and n+2 are compared. Because these two pixels will have the same chrominance phase, the difference between the pixels can be directly compared with a noise threshold.

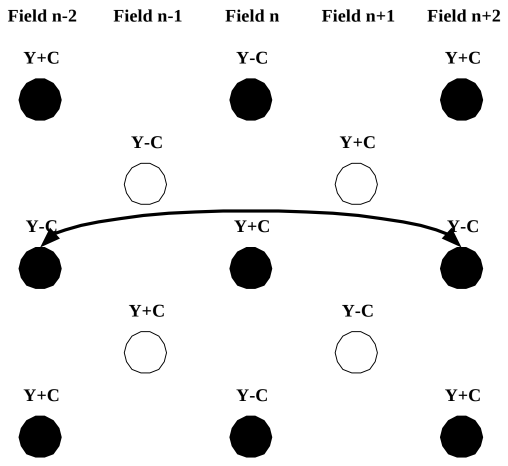


Figure 12 – Small motion detection

Small motion detection is depicted in figure 13. Here, only the luminance information is used, and the difference is again compared with a noise threshold.

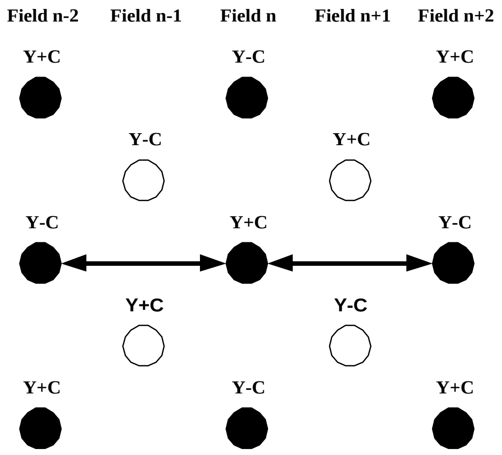


Figure 13 – Small motion detection

The results from two methods are then averaged and compared against a motion threshold: Pixel Motion = (Large Motion detector + Small Motion detector) /2

7. Color demodulation

The chrominance demodulator accepts modulated chroma data from the Y/C separator, and the color burst information that was extracted before. It generates the U and V color information. During active video, the chrominance data is demodulated using sine and cosine sub-carrier data as shown in figure 14.

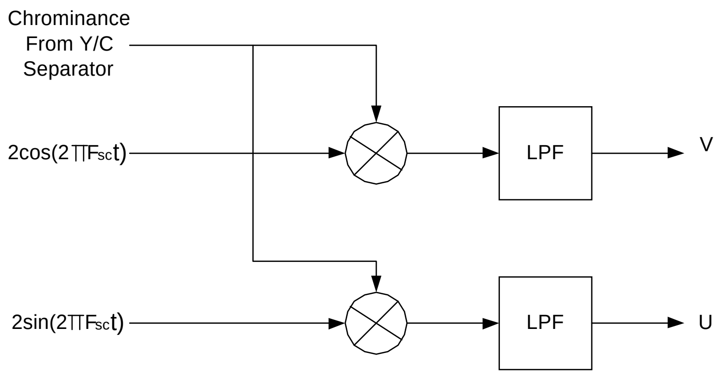


Figure 14 - Color demodulation

The chrominance signal processed by the demodulator may be represented by:

$$Chroma = (U \sin \omega t) + (V \cos \omega t)$$

U is obtained by multiplying the chrominance data by $(2 \sin \omega t)$.
V is obtained by multiplying the chrominance data by $(2 \cos \omega t)$.

$$[(U \sin \omega t) + (V \cos \omega t)](2 \sin \omega t) = U - (U \cos 2\omega t) + (V \sin 2\omega t)$$
$$[(U \sin \omega t) + (V \cos \omega t)](2 \cos \omega t) = V - (V \cos 2\omega t) + (U \sin 2\omega t)$$

The double frequency components are then removed by low-pass filtering, resulting in the U and V signals being recovered.

8. De-interlacer

In traditional NTSC TV systems, the display is interlaced at a rate of 60 fields/second. The vertical resolution is 480 lines, but only 240 scan lines are displayed with each field and the total frame rate is 30 frames/second.
The De-interlacer converts this interlaced 60 fields/second sequence with half resolution into a progressive 60 frames/second with full resolution.
In this project, a motion adaptive de-interlacer was implemented. The motion information is taken from the same motion detector that was developed for the 3D Y/C separation.
A block diagram of the de-interlacer is depicted in figure 15.

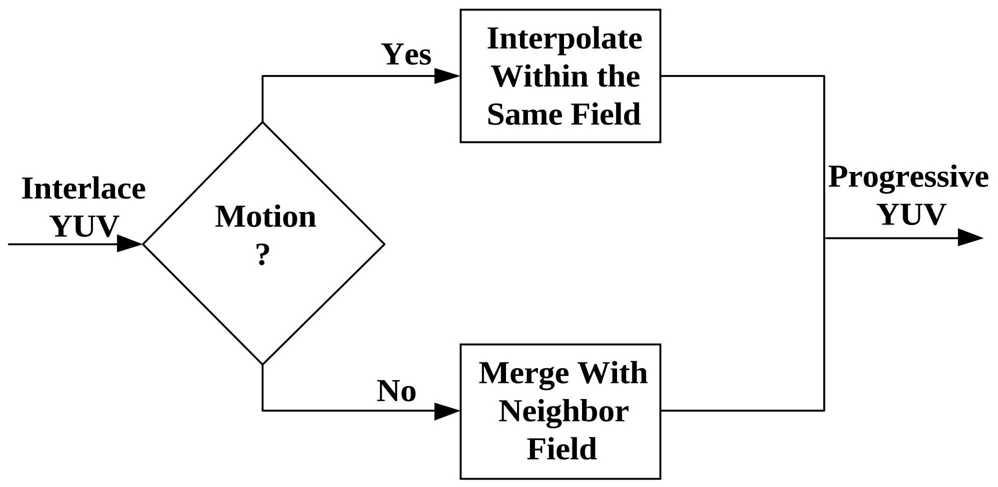


Figure 15 - De-interlacer operation

Working in the pixel domain, in areas of the image where motion is present, the image is interpolated. In static areas the image is weaved. The interpolation for the missing pixel is done by averaging the pixels in the adjacent lines within the same field. The weave operation is done by simply copying the missing pixel from the same pixel location in the adjacent field.

9. YUV to RGB color space conversion

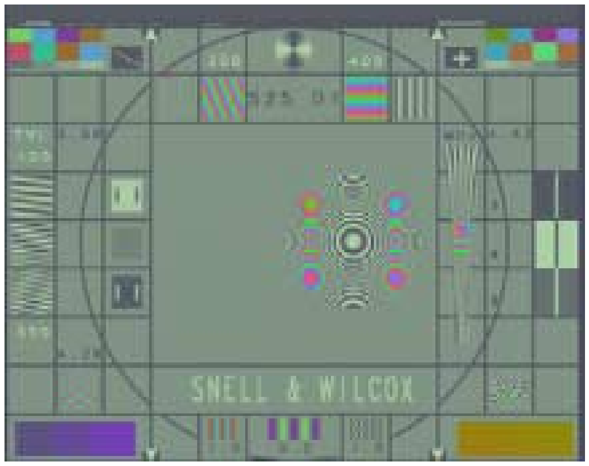
To display the progressive information on a PC monitor, a conversion from a YUV color space to RGB color space is required.
The YUV information was prepared in a 4D matrix such that each 3D matrix holds the YUV information for each frame.
The following conversion equations were used to convert to RGB:

$$R = Y + 1.140V$$
$$G = Y - 0.395U - 0.581V$$
$$B = Y + 2.032U$$

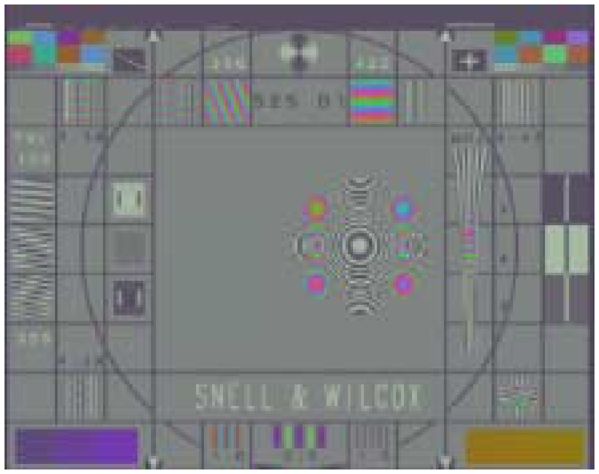
10. Additional experimental results

The implemented video processing was tested on several test patterns and motion sequences. Following are selected decoded frames in progressive resolution and RGB color space.
Figure 16 is the Snell & Wilcox challenging test sequence. In this sequence, the round plate is the only object moving in the scene, and in different areas of the picture there are patterns of high frequency luminance with vertical, diagonal and horizontal information. The figure shows the same frame decoded using notch filter, 2D adaptive filter and 3D adaptive filter for the Y/C separation. It can be easily observed that the use of notch filter results in cross-luminance and cross-chrominance and lost of luminance high frequencies. The 2D comb filter performs slightly better, but it still has a lot of artifacts that mainly caused by a wrong decision of the edge detection to switch to notch filter.
The 3D comb filter performs much better but still has artifacts that happen due to wrong motion detection in the diagonal high frequencies areas.

SNELL & WILCOX WITH NOTCH FILTER



SNELL & WILCOX 2D ADAPTIVE COMB FILTER



SNELL & WILCOX WITH ADAPTIVE 3D COMB FILTER

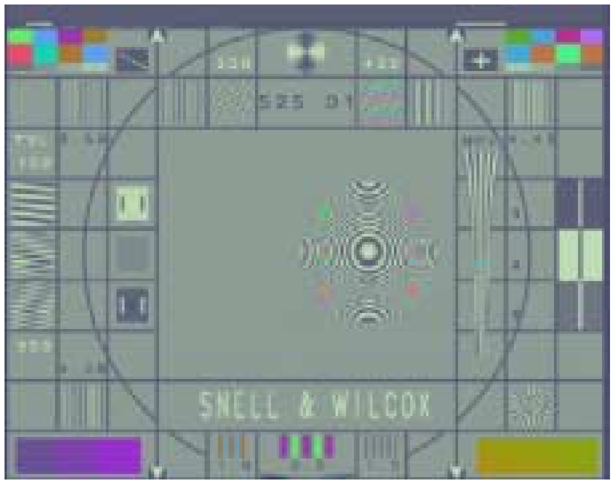
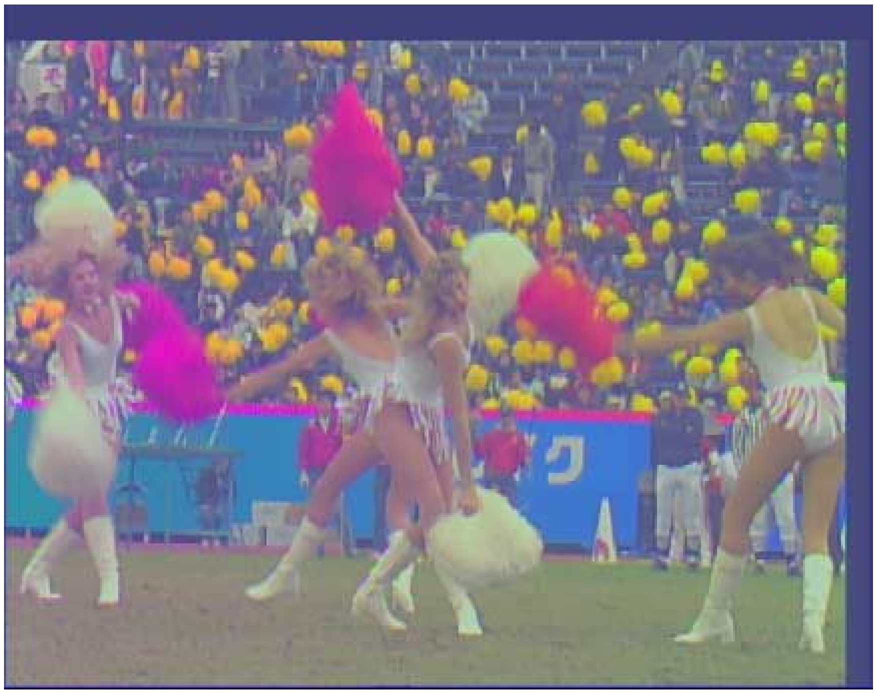


Figure 16 - Snell & Wilcox test pattern decoded with 3 different Y/C separation methods

Figure 17 is a frame from the “Cheer Leader” sequence decoded using 3D adaptive comb filter. In this sequence, there is a lot of motion in the picture, and some of the motions are fairly large (cheer leader hands). The frame was decoded correctly and there are hardly any motion artifacts.

"CHEER LEADER" WITH 3D ADAPTIVE COMB FILTER



11. Conclusions

In this project, a full implementation for digital processing of analog TV signal was accomplished. It is clear that digital processing of the analog composite video enables the usage of different algorithms that cannot be implemented in the analog domain. The video quality can be highly improved and decoding can be easier and more robust.
The following was observed during implementation and testing:

- The Y/C separation as well as the de-interlacer performance is highly depended on the motion detection accuracy. The motion detection algorithm can be improved with the expense of memory usage and computations.
- Some of the Y/C separation artifacts cannot be removed with the 3D adaptive comb filter. For example, high spatial frequency moving objects.
- For accurate color demodulation, a good color burst phase lock loop is required.

12. References

[1] Video Demystified, Keith Jack, Third edition
[2] The Art of Digital Video, John Watkinson
[3] Video Processing and Communications, Yao Wang. Chapters 1,3,4
[4] N.M. Tekalp, Digital Video Processing, Prentice Hall, 1995

Apendix – Matlab code

```
function [video_mat,color_burst_mat]=video_extract(raw_video)

%**********************************************************************
% Written by Lior Zimet March 2002
%
% THIS FUNCTION EXTRACT THE ACTIVE VIDEO AND THE COLOR BURST INFORMATION
% FROM A BINARY FILE OF 27MHZ SAMPLED VIDEO.
%
% INPUT - RAW VIDEO BINARY FILE
%
% OUTPUT - 3D MATRIX OF ACTIVE VIDEO. EACH 2D MATRIX IS ONE FIELD
%               EACH RAW IS ONE VIDEO LINE
%          THE COLOR BURST MAT CONTAINS THE FREQUENCY AND PHASE IN EACH
%          VIDEO LINE
%**********************************************************************

% SAMPLING RATE IS 27MHZ MEANING 37ns PERIOD.
[m,n]=size(raw_video);
sync_threshold=800;       %threshold for sync detection
vsync_period_threshold=700;
hsync_period_threshold=100;

% First vertical sync detection

vsync_flag=0;
hsync_flag=0;
vsync_progress=0;
vsync_counter=0;
hsync_counter=0;

for i=1:m
   if raw_video(i) < sync_threshold
      vsync_counter=vsync_counter+1;
      if vsync_counter > vsync_period_threshold
         vsync_flag=1;
      end
   else
      if vsync_flag==1
         vsync_progress=vsync_progress+1;
         vsync_flag=0;
         vsync_counter=0;
         if vsync_progress == 6
            end_of_1st_vsync=i
            vsync_progress=0;
            break;
         end
      end
   end
end
```

```
% From end of VSYNC, detect every HSYNC and store active video field
% from video lines 10-262 for field0 and 272-525 for field1

% End of VSYNC is at the end of line 6
% Lines 7,8,9 has equalizing pulses and HSYNC pulse every half line period

active_pxl_per_line= 1440;
line_number=0;
field_number=1;
hsync_low2active_video=round(9.2/0.037);        %active video is 9.2us from falling
edge of HSYNC
hsync_low2color_burst=round(5.3/0.037);         %color   burst   starts   5.3us   from
falling edge of HSYNC
color_burst_length=round(10*(1/3.579545e6)/37e-9);%color   burst   length   is   9+-1
cycles of 3.579545MHZ

i=end_of_1st_vsync;
while i<=m
   if raw_video(i) < sync_threshold
      hsync_counter=hsync_counter+1;
      vsync_counter=vsync_counter+1;
      if vsync_counter > vsync_period_threshold
         vsync_flag=1;
      end
      if    (hsync_counter    >    hsync_period_threshold    &    hsync_counter    <
vsync_period_threshold)
         hsync_flag=1;
      else
         hsync_flag=0;
      end
   else
      % HSYNC case
      if hsync_flag == 1
         line_number=line_number+1;
         start_of_active_vid=i-hsync_counter+hsync_low2active_video;
         video_mat(line_number,:,field_number)=...
            raw_video(start_of_active_vid:start_of_active_vid+active_pxl_per_line-
1);

         %Extracting the line color burst frequency
         start_of_color_burst=i-hsync_counter+hsync_low2color_burst;

color_burst=raw_video(start_of_color_burst:start_of_color_burst+color_burst_length)
;
         cb_interp=interp(color_burst,100)-color_burst(1);
         cb_sign=sign(cb_interp)+1;
         cb_zero_cross=diff(cb_sign);
         [zero_index,s]=find(cb_zero_cross>0);
         color_burst_mat(line_number,1,field_number)=1/(mean(diff(zero_index(4:end-
4)))*37e-11);
         %color burst phase 1=0 degrees -1=180 degrees
         color_burst_mat(line_number,2,field_number)=cb_sign(700)/2;

         %reseting counters and progress i to the next line
         i=start_of_active_vid+active_pxl_per_line-1;
         vsync_counter=0;
         hsync_counter=0;
         hsync_flag=0;
      % VSYNC case
      elseif vsync_flag==1
         vsync_progress=vsync_progress+1;
         vsync_flag=0;
         vsync_counter=0;
         hsync_flag=0;
         hsync_counter=0;
         if vsync_progress == 6
            line_number=0;
            field_number=field_number+1
            vsync_progress=0;
         end
      % Equalizing pulses case
      else
         vsync_counter=0;
         hsync_counter=0;
      end
   end
   i=i+1;
end
```

```
function [Y, C,MD_mat]=YC_seperation(video_mat)


%***********************************************************************
% Written by Lior Zimet March 2002

% THIS FUNCTION Y/C SEPARATION USING 3D ADAPTIVE METHODE
% IT INCLUDES RESAMPLER TO 4x Fsc, NOTCH FILTER, 2D ADAPTIVE,
% 3D COMB FILTER, AND MOTION DETECTION
%
% INPUT - 3D FIELDS MATRIX OF ACTIVE VIDEO
%
% OUTPUT - 3D MATRIX OF LUMINANCE
%               3D MATRIX OF CHROMINANCE
%          MOTION DETECTION MATRIX TO BE USED LATER BY THE DE-INTERLACER
%
%************************************************************************


[m,n,field]=size(video_mat);
[b,a] = butter(24,[2.28 4.88]/7.16,'stop');
y_edge_th=300;
c_edge_th=500;
noise_th=50;

%resample to 4*Fsc (T=69.8ns)
for f=1:field
      video_mat_4Fsc(:,:,f)=resample(video_mat(:,:,f)',35,66)';
end

clear video_mat;

[m,n,field]=size(video_mat_4Fsc);

% 1D and 2D Y/C seperation
for f=1:field
   % first and last line of each field
   Y1D(1,:,f)=(video_mat_4Fsc(1,:,f)+video_mat_4Fsc(2,:,f))/2;
   Y1D(m,:,f)=(video_mat_4Fsc(m-1,:,f)+video_mat_4Fsc(m,:,f))/2;
   C1D(1,:,f)=(video_mat_4Fsc(1,:,f)-video_mat_4Fsc(2,:,f))/2;
   C1D(m,:,f)=(video_mat_4Fsc(m-1,:,f)-video_mat_4Fsc(m,:,f))/2;


   Y_notch(1,:,f)=filtfilt(b,a,video_mat_4Fsc(1,:,f));
   Y_notch(2,:,f)=filtfilt(b,a,video_mat_4Fsc(2,:,f));
   C_notch(1,:,f)=video_mat_4Fsc(1,:,f)-Y_notch(1,:,f);
        C_notch(2,:,f)=video_mat_4Fsc(2,:,f)-Y_notch(2,:,f);

   for i=2:m-1
      % Notch filter
      Y_notch(i+1,:,f)=filtfilt(b,a,video_mat_4Fsc(i+1,:,f));
      C_notch(i+1,:,f)=video_mat_4Fsc(i+1,:,f)-Y_notch(i+1,:,f);
           % 1D Y/C seperation - vertically
      Y1D1=(video_mat_4Fsc(i,:,f)+video_mat_4Fsc(i-1,:,f))/2;
      Y1D2=(video_mat_4Fsc(i,:,f)+video_mat_4Fsc(i+1,:,f))/2;

      C1D1=(video_mat_4Fsc(i,:,f)-video_mat_4Fsc(i-1,:,f))/2;
      C1D2=(video_mat_4Fsc(i,:,f)-video_mat_4Fsc(i+1,:,f))/2;

      % adaptive 2D for C
      pc1=abs(C_notch(i-1,:,f)+C_notch(i,:,f))/2;
      pc2=abs(C_notch(i,:,f)+C_notch(i+1,:,f))/2;
      for j=1:n
         if (pc1(j)==0 & pc2(j)==0)
            ac1=0.5;
            ac2=0.5;
            C1D(i,j,f)=ac1*C1D1(j)+ac2*C1D2(j);
        elseif (pc1(j)==0 & pc2(j)~=0)
           ac1=1;
           ac2=0;
           C1D(i,j,f)=ac1*C1D1(j)+ac2*C1D2(j);
              elseif (pc1(j)~=0 & pc2(j)==0)
              ac1=0;
              ac2=1;
           C1D(i,j,f)=ac1*C1D1(j)+ac2*C1D2(j);
        elseif (pc1(j)>c_edge_th & pc2(j)>c_edge_th)
           C1D(i,j,f)=C_notch(i,j,f);
              else
                 ac1=pc2(j)/(pc1(j)+pc2(j));
                 ac2=pc1(j)/(pc1(j)+pc2(j));
           C1D(i,j,f)=ac1*C1D1(j)+ac2*C1D2(j);
         end
      end

      % adaptive 2D for Y
      py1=abs(Y_notch(i-1,:,f)-Y_notch(i,:,f))/2;
      py2=abs(Y_notch(i,:,f)-Y_notch(i+1,:,f))/2;
      for j=1:n
         if (py1(j)==0 & py2(j)==0)
            ay1=0.5;
            ay2=0.5;
            Y1D(i,j,f)=ay1*Y1D1(j)+ay2*Y1D2(j);
        elseif (py1(j)==0 & py2(j)~=0)
           ay1=1;
           ay2=0;
           Y1D(i,j,f)=ay1*Y1D1(j)+ay2*Y1D2(j);
              elseif (py1(j)~=0 & py2(j)==0)
              ay1=0;
              ay2=1;
           Y1D(i,j,f)=ay1*Y1D1(j)+ay2*Y1D2(j);
        elseif (py1(j)>y_edge_th & py2(j)>y_edge_th)
           Y1D(i,j,f)=Y_notch(i,j,f);
              else
              ay1=py2(j)/(py1(j)+py2(j));
           ay2=py1(j)/(py1(j)+py2(j));
           Y1D(i,j,f)=ay1*Y1D1(j)+ay2*Y1D2(j);
         end
      end
   end
end

% 3D Y/C seperation - temporally

Y3D(:,:,1)=(video_mat_4Fsc(:,:,1)+video_mat_4Fsc(:,:,3))/2;
% first field has no previous field
```

```
Y3D(:,:,field)=(video_mat_4Fsc(:,:,field)+video_mat_4Fsc(:,:,field-2))/2;
% last field has no next field
Y3D(:,:,2)=(video_mat_4Fsc(:,:,2)+video_mat_4Fsc(:,:,4))/2;
% first field has no previous field
Y3D(:,:,field-1)=(video_mat_4Fsc(:,:,field-1)+video_mat_4Fsc(:,:,field-3))/2;
% last field has no next field
C3D(:,:,1)=video_mat_4Fsc(:,:,1)-Y3D(:,:,1);
C3D(:,:,2)=video_mat_4Fsc(:,:,2)-Y3D(:,:,2);
C3D(:,:,field)=video_mat_4Fsc(:,:,field)-Y3D(:,:,field);
C3D(:,:,field-1)=video_mat_4Fsc(:,:,field-1)-Y3D(:,:,field-1);

for f=3:field-2

Y3D(:,:,f)=(video_mat_4Fsc(:,:,f)+video_mat_4Fsc(:,:,f+2)+video_mat_4Fsc(:,:,f)+vid
eo_mat_4Fsc(:,:,f-1))/4;
       C3D(:,:,f)=video_mat_4Fsc(:,:,f)-Y3D(:,:,f);
end


% Motion detection and Y/C adaptive decision
LF_sad=zeros(m,n,field);
HF_sad=zeros(m,n,field);
%HF_sad(:,:,1:2)=100;

LF_sad(:,:,1)=abs(diff(Y_notch(:,:,1:2:3),1,3));
LF_sad(:,:,2:2:end-1)=abs(diff(Y_notch(:,:,1:2:end),1,3));
LF_sad(:,:,3:2:end)=abs(diff(Y_notch(:,:,2:2:end),1,3));

for f=1:field-3
   LF_sad(:,:,f)=(LF_sad(:,:,f)+LF_sad(:,:,f+2))/2;
   LF_sad(:,:,f+1)=(LF_sad(:,:,f+1)+LF_sad(:,:,f+3))/2;
end
   LF_sad(:,:,field-2)=(LF_sad(:,:,field-2)+LF_sad(:,:,field))/2;
   LF_sad(:,:,field-1)=(LF_sad(:,:,field-1)+LF_sad(:,:,field-3))/2;
   LF_sad(:,:,field)=(LF_sad(:,:,field)+LF_sad(:,:,field-2))/2;


for f=3:field-2
   HF_sad(:,:,f)=abs(video_mat_4Fsc(:,:,f-2)-video_mat_4Fsc(:,:,f+2));
end
MD_mat=((LF_sad+HF_sad)/2>noise_th);

clear video_mat_4Fsc;
clear Y_notch;
clear LF_sad;
clear HF_sad;

Y=Y3D;
C=C3D;
Y(MD_mat==1)=Y1D(MD_mat==1);
C(MD_mat==1)=C1D(MD_mat==1);
```

```
function [YUV_field]=chroma_demodulator(Y,C,color_burst)

%**********************************************************************
% Written by Lior Zimet March 2002

% THIS FUNCTION PERFORMS THE COLOR DEMODULATION
%
% INPUT - 3D LUMINANCE AND CHROMINANCE MATRICES.
%         COLOR BURST INFORMATION MATRIX
%
% OUTPUT - 4D MATRIX OF YUV FIELDS
%
%***********************************************************************

[m,n,field]=size(C);
sample_time=69.8e-9;            %4*Fsc sampling period
[b,a] = butter(12,0.6/(3.58));   %filter coefficients for LPF
YUV_field=zeros(m,n,3,field);

for f=1:field
   YUV_field(:,:,1,f)=Y(:,:,f)*(255/max(max(Y(:,:,f))));
end
clear Y;

for f=1:field
   for l=1:m
      U(l,:)=filtfilt(b,a,2*sin(2*pi*color_burst(l,1,f)*(0:sample_time:(n-
1)*sample_time)+color_burst(l,2,f)*pi+pi).*C(l,:,f));
      V(l,:)=filtfilt(b,a,2*cos(2*pi*color_burst(l,1,f)*(0:sample_time:(n-
1)*sample_time)+color_burst(l,2,f)*pi+pi).*C(l,:,f));
   end

   YUV_field(:,:,2,f)=U*(112/max(max(U)));
   YUV_field(:,:,3,f)=V*(157/max(max(V)));

end
```

```
function [RGB_frame]=deint(YUV_field,MD_mat)

%**********************************************************************
% Written by Lior Zimet March 2002
%
% THIS FUNCTION IMPLEMENTS INTERLACE TO PROGRESSIVE CONVERSION
% AND YUV TO RGB COLOR SPACE CONVERSION
%
% INPUT - 4D MATRIX OF YUV FIELDS
%
% OUTPUT - 4D MATRIX OF RGB FRAMES
%
%***********************************************************************

[m,n,k,field]=size(YUV_field);

YUV_frame=zeros(2*m,n,3,field);

%preparing matrices with double vertical resolution
YUV_frame(1:2:end,:,:,1:2:field)=YUV_field(:,:,:,1:2:field);
YUV_frame(2:2:end,:,:,2:2:field)=YUV_field(:,:,:,2:2:field);

clear YUV_field;

for f=1:2:field-1
   for i=2:m
      for j=1:n

         %no motion detected for this pixel
         if MD_mat(i,j,f+1)==0
            YUV_frame(i*2,j,:,f)=YUV_frame(i*2,j,:,f+1);
         % motion detected --> interpolate vertically
         else
            YUV_frame(i*2,j,:,f)=(YUV_frame(i*2-1,j,:,f)+YUV_frame(i*2+1,j,:,f))/2;
         end

         %no motion detected for this pixel
         if MD_mat(i,j,f)==0
            YUV_frame(i*2-1,j,:,f+1)=YUV_frame(i*2-1,j,:,f);
         % motion detected --> interpolate vertically
         else
            YUV_frame(i*2-1,j,:,f+1)=(YUV_frame(i*2-
2,j,:,f+1)+YUV_frame(i*2,j,:,f+1))/2;
         end

      end
   end
end




%YUV to RGB conversion
```

```
[m,n,k,frame]=size(YUV_frame);
conversion_matrix=[1.000 0.000 1.140; 1.000 -0.395 -0.581; 1.000 2.032 0.001];
for f=1:frame
   for i=1:m
      for j=1:n
         yuv_set=[YUV_frame(i,j,1,f),YUV_frame(i,j,2,f),YUV_frame(i,j,3,f)]';
         RGB_frame(i,j,1:3,f)=conversion_matrix * yuv_set;
      end
   end
   %Prepare the RGB matrices
   if min(min(RGB_frame(:,:,1,f)))<0
      RGB_frame(:,:,1,f)=RGB_frame(:,:,1,f)+abs(min(min(RGB_frame(:,:,1,f))));
   end
   if min(min(RGB_frame(:,:,2,f)))<0
      RGB_frame(:,:,2,f)=RGB_frame(:,:,2,f)+abs(min(min(RGB_frame(:,:,2,f))));
   end
   if min(min(RGB_frame(:,:,3,f)))<0
      RGB_frame(:,:,3,f)=RGB_frame(:,:,3,f)+abs(min(min(RGB_frame(:,:,3,f))));
   end
   RGB_frame(:,:,1,f)=RGB_frame(:,:,1,f)*(1/max(max(RGB_frame(:,:,1,f))));
   RGB_frame(:,:,2,f)=RGB_frame(:,:,2,f)*(1/max(max(RGB_frame(:,:,2,f))));
   RGB_frame(:,:,3,f)=RGB_frame(:,:,3,f)*(1/max(max(RGB_frame(:,:,3,f))));
end
```