[Level 2 Project Topic]
Implement an error concealment algorithm that recovers the damaged areas at the decoder based on characteristics of image and video signals. You can compare the performance of spatial, temporal, and hybrid error concealment algorithms in terms of performance, reconstructed picture quality, and computational complexity.

Error Concealment in Encoded Video Streams

ECE 508 – Video Communications
(Fall 2013)

Andrea Sancho Silgado (CWID: A20315328)

INTRODUCTION

Although the continuous improvement in reducing errors in data transmission, we know that we cannot avoid completely the loss of information in our most used networks. We can rely on existing detection and correction techniques in coding, but when delay is highly relevant (Video Conferences/Calls, Video Streaming…) in real time video, we cannot afford retransmission requests. Then, mechanisms as forward error correction (FEC) and automatic retransmission request (ARQ) are not suitable for this type of transmissions. The first one requires higher bandwidth and the second one results in prohibitive transmission delays.

The process of error concealment consists in estimating or replacing lost information in the transmission. There are two main technologies for achieving this goal:

- The spatial error concealment, which uses the inherent correlation in the spatial domain.
- The temporal error concealment, which uses the correlation between frames of other instants.

For this project I have implemented existing spatial and temporal error concealment algorithms and proposed alternative designs.

DESIGN

Goals

The main goal of the project is the implementation of error concealment algorithms in Matlab. As starting point I will use two consecutive frames of a video sequence. I will assume that after the decoding we have identified a block loss/error, involving the loss of pixels in one of the frames being analyzed. For those frames, then, we will apply:

- **Spatial Error Concealment**: especially important for the first transmitted I-frames, where the temporal information is not available. For the implementation I will use the linear interpolation technique consisting in taking the weighted average of the surrounding pixels to obtain an approximation of the missing information. (*References [6] [7] [8]*)

- **Temporal Error Concealment**: for this I will use the information of previous frames. It is the equivalent of the process developed in Block Matching Algorithms, where we look for the best match among a search range in the frame. (*References [9])*

- **Spatial-Temporal Error Concealment**: I will try to implement a successful combination of both techniques to improve the quality of the image. (*References [10] [11] [12]*)

After implementing those algorithms, I will carry out the comparison of the performance of each error concealment scheme taking into account the frame quality (calculating PSNR and comparing visually the frames), and the computational complexity (processing times).

Work Timeline

Date	Time	Work
10/20	5 days	- Gather information about the topic - Obtain reference papers
10/25	5 days	- Describe objectives and project goals - Project proposal
10/30	**PROJECT PROPOSAL DUE**	
10/30	1 week	- Implementation of Spatial Error Concealment Algorithm - Test the designed algorithm - Collect results - Evaluate the performance
11/06	1 week	- Implementation of Temporal Error Concealment Algorithm - Test the designed algorithm - Collect results - Evaluate the performance
11/13	10 days	- Implementation of Hybrid Error Concealment Algorithm - Test the designed algorithm - Collect results - Evaluate the performance
11/24	1 week	- Compare Error Concealment Schemes - Analyze results obtained
12/01	5 days	- Finish final report
12/05	**FINAL PROJECT DUE**	

* If time permits, I will try to implement a secondary Spatial Error Concealment Algorithm based on the directional interpolation during the week assigned to this part.

Simulate Loss of Information

The first step in the process is to simulate the block loss. To do this I implemented a code for generating equidistant missing blocks in vertical and horizontal directions. To set up the desired concentration of the blocks in each axis, you can just modify this values in the code:

```
% Setup block error parameters:
concentrationVertical = 4;
concentrationHorizontal = 4;
```

Then the configured values are used to create two vectors (`verticalLoss` and `horizontalLoss`) which contain the position (vertical and horizontal respectively) of the upper left pixel of each missing block. Then, those blocks are set to black (value 0 in grey scale).

To easily test the proposed error concealment techniques, with this pattern it is assured that there are always four neighboring blocks for a lost block.

Spatial Error Concealment

- **Linear Interpolation**: **[SpatialEC_linear.m]**

In the implementation I used the information provided in class. Here we scan the entire frame looking for the position of the missing blocks referring it to the location of its upper left pixel. Then, for each missing pixel in the lost block, we search the pixels in the received blocks that will be used for the estimation:

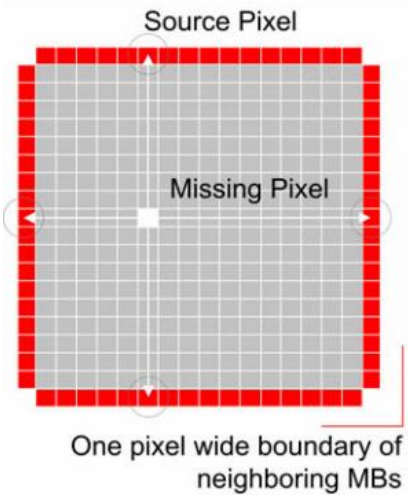


Figure from Reference [7]

The red pixels are the ones belonging to the four neighbor blocks (north, south, east and west).

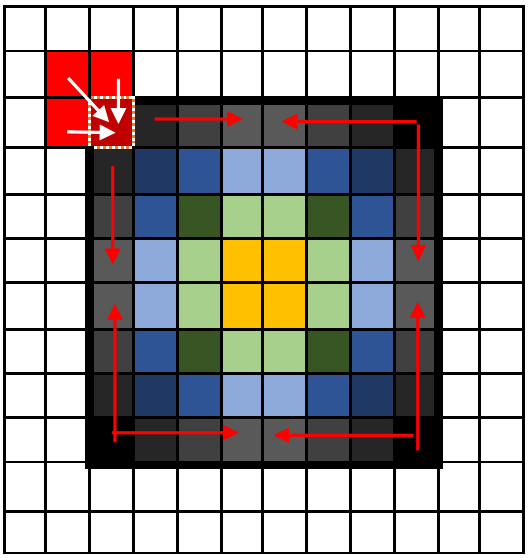
Applying the linear interpolation formula, we can obtain the estimation:

$$B(o1+i,o2+j)=\left[\frac{1}{d_N+d_S+d_E+d_W}\right]\times[(d_N\times B_S(o1+blockSize+1,o2+j))+(d_S\times B_N(o1,o2+j))+(d_E\times B_W(o1+i,o2))+(d_W\times B_E(o1+i,o2+blockSize+1))]$$

The values o1 and o2 give us the upper-left reference position of the block. As variables i and j belong to the range (1,blockSize), we can go through every pixel in the block.

- **Pseudo-Directional Interpolation**: [SpatialEC.m]

I implemented this algorithm with the following idea:



1. For each orientation (North, South, East and West) we first estimate the corners of the missing block. For each corner-pixel we use and 'L' type estimation this means that we use the average (multiplying each value by a weight of 1/3) of three pixels in neighbor blocks as shown in the image (white arrows in red pixels).

2. After we continue in each direction, taking as origin the corners. As the red arrows show in the image.

3. We first complete the most external frame (the black and grey one) and then restart the process for the next frame of pixels (the blue one, then the green, and finally the yellow one).

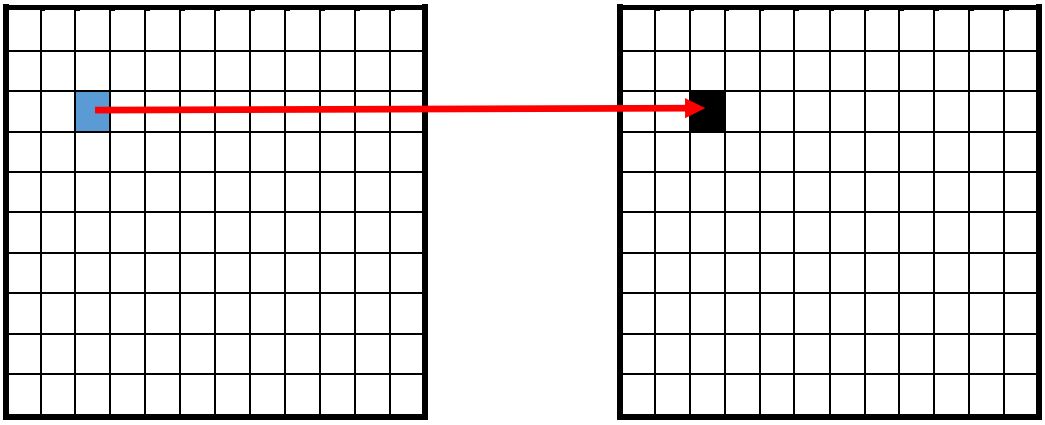
In order to compute the process I needed to divide the process into four parts: North, South, East and West, and for each of them I divided the loop into two, one per direction. So each red arrow will represent a loop in the code. As it is directly observed in the explanatory image, for the next frames, the inner ones, (blue, green…) the number of operation is decreased as we approximate to the center of the block.

Also, we have to notice that with this implementation the only pixels estimated with 'correct pixels' are the ones in the corners. The rest of the missing pixels in the block use, at least, a previously estimated pixel.

Temporal Error Concealment

- **Zero-Motion Vector**: [TemporalEC_zeromotion.m]

This method is the simplest of all. To implement the estimation of a missing block we make the assumption that the previous frame was received correctly. Then we simply look for the block in the same position in the previous frame and copy it in the missing block of the present frame.



Previous Received Frame

Frame with the missing blocks

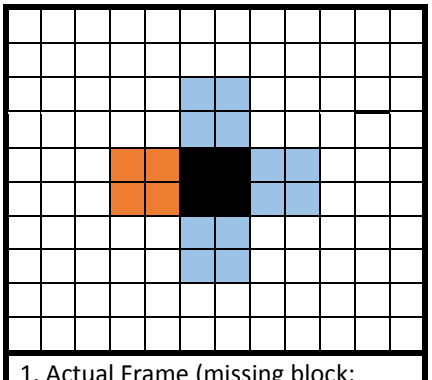
- **Block-Matching**: **[TemporalEC.m]**

The implementation of this method is based on the EBMA structure. As we have a missing block in the received frame, we identify its four block neighbors (the north, south, east and west ones). Applying the EBMA technique, I try to locate each neighbor in the previous frame.

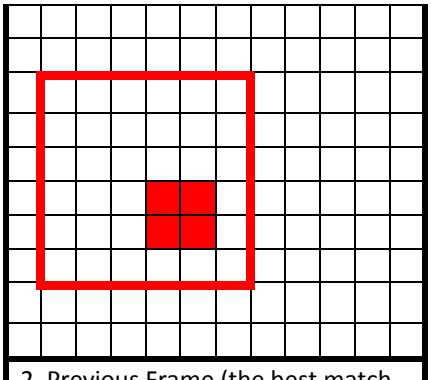
As it happened in the EBMA algorithm, seeking the best match through the entire image, with pixel accuracy, requires a lot of computational time so I limited the search range to a low fixed value (`range = 14`).

In the algorithm I start with the north neighbor, then after finding the best match for this block in the previous frame we update the values: `k = a; dy = m; dx = n.` After performing the search for each neighbor block as result we will have the least value obtained of 'k', in other words, the best block match that will be used to determine the missing block.

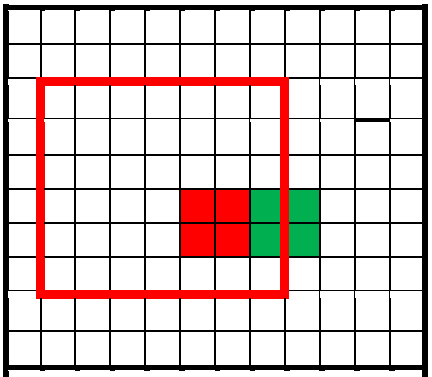
EXAMPLE: if in the search we determine that the west neighbor got the least difference respect the block found in the previous frame, then the immediate block at its right side will be the one copied in the missing block of our actual frame.



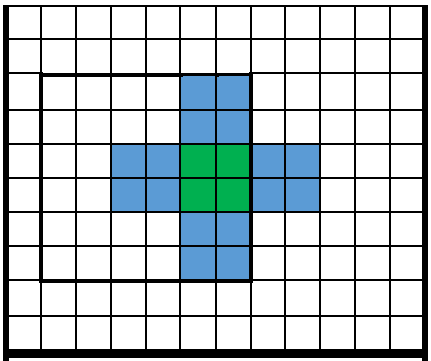
1. Actual Frame (missing block: black) (best block match: west neighbor)



2. Previous Frame (the best match for the west neighbor block found in the search range)



3. Previous Frame: Locate the estimated missing block based on the founded neighbor.



4. Actual Frame (copy the block in the missing area)

Hybrid Error Concealment

- **Spatial-Temporal Error Cocnealment: [HybridEC.m]**

After the implementation of the spatial and temporal algorithms, to design the hybrid method, I combined the best of the previous (one of each type) and with a simple averaging of the results, obtain the best performance.

For the spatial technique, the best result was achieved by the linear interpolation code, and regarding the temporal technique the best result was achieved by the block-matching technique.

I performed tests assigning different weights to each technique, and the best result was obtained when the spatial technique weighted 0.6 and the temporal 0.4 for the estimated block.

RESULTS

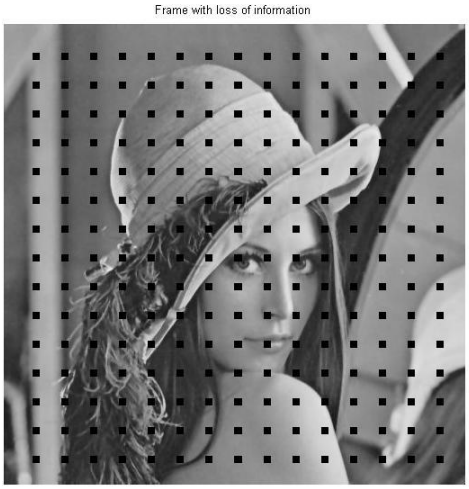
Expected results

For the spatial error concealment I expect a simple, low-computational algorithm. It should result in the fastest of the three proposed ones. For the temporal error concealment, the complexity should be similar to the EBMA project already done. I expect to obtain a higher quality although the delay should also be higher due to the computational level in comparison with the first one.

For the last part, as it should include the advantages of both algorithms, I expect it to achieve the best performance, but also the complexity will be the highest. Quality expected for the latter should be the best.

Obtained results

- **Linear Interpolation (1): [SpatialEC_linear.m]**



- Processing time in spatial error concealment = 0.4524 seconds
- PSNR without error concealment = 18.3126 dB
- PSNR with error concealment = 38.5948 dB
- PSNR gain = 20.2822 dB

➢ **Pseudo-Directional Interpolation (1): [SpatialEC.m]**

- Processing time in spatial error concealment = 0.4992 seconds
- PSNR without error concealment = 18.3126 dB
- PSNR with error concealment = 36.3461 dB
- PSNR gain = 18.0335 dB

➢ **Linear Interpolation (2): [SpatialEC_linear.m]**

- Processing time in spatial error concealment = 0.4524 seconds
- PSNR without error concealment = 19.1003 dB
- PSNR with error concealment = 35.8406 dB
- PSNR gain = 16.7403 dB

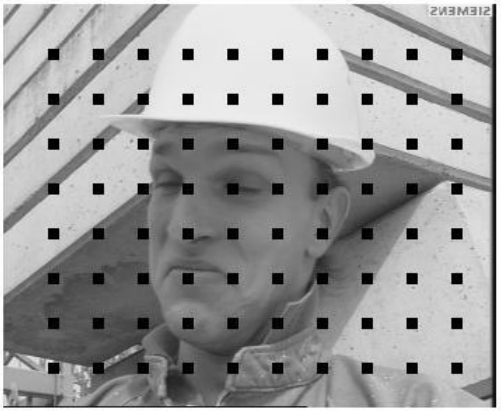
➢ **Pseudo-Directional Interpolation (2):** **[SpatialEC.m]**

- Processing time in spatial error concealment = 0.4992 seconds
- PSNR without error concealment = 19.1003 dB
- PSNR with error concealment = 32.3383 dB
- PSNR gain = 13.238 dB

- **Linear Interpolation (3)**: **[SpatialEC_linear.m]**

Original frame

Frame with loss of information



Frame with spatial error concealment

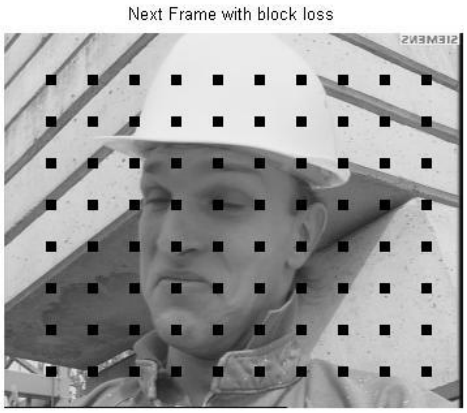
- Processing time in spatial error concealment = 0.4056 seconds
- PSNR without error concealment = 16.7951 dB
- PSNR with error concealment = 37.786 dB
- PSNR gain = 20.9909

- **Pseudo-Directional Interpolation (3)**: **[SpatialEC.m]**

Frame with spatial error concealment

- Processing time in spatial error concealment = 0.4524 seconds
- PSNR without error concealment = 16.7951 dB
- PSNR with error concealment = 35.7054 dB
- PSNR gain = 18.9103 dB

- **Zero-Motion Vector**: **[TemporalEC_zeromotion.m]**

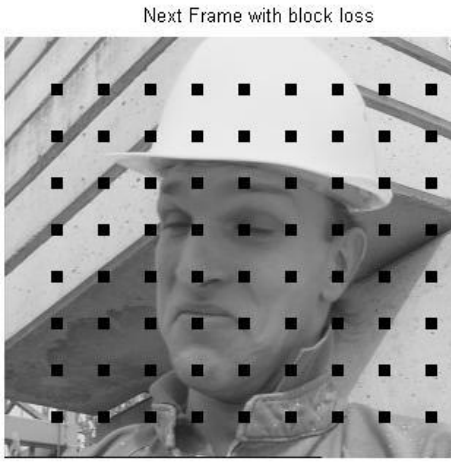


- Processing time in temporal error concealment = 0.6864 seconds
- PSNR without error concealment = 16.7951 dB
- PSNR with error concealment = 32.0223 dB
- PSNR gain = 15.2272 dB

- **Block-Matching**: **[TemporalEC.m]**

- Processing time in temporal error concealment = 4.6176 seconds
- PSNR without error concealment = 16.7951 dB
- PSNR with error concealment = 37.5822 dB
- PSNR gain = 20.7871 dB

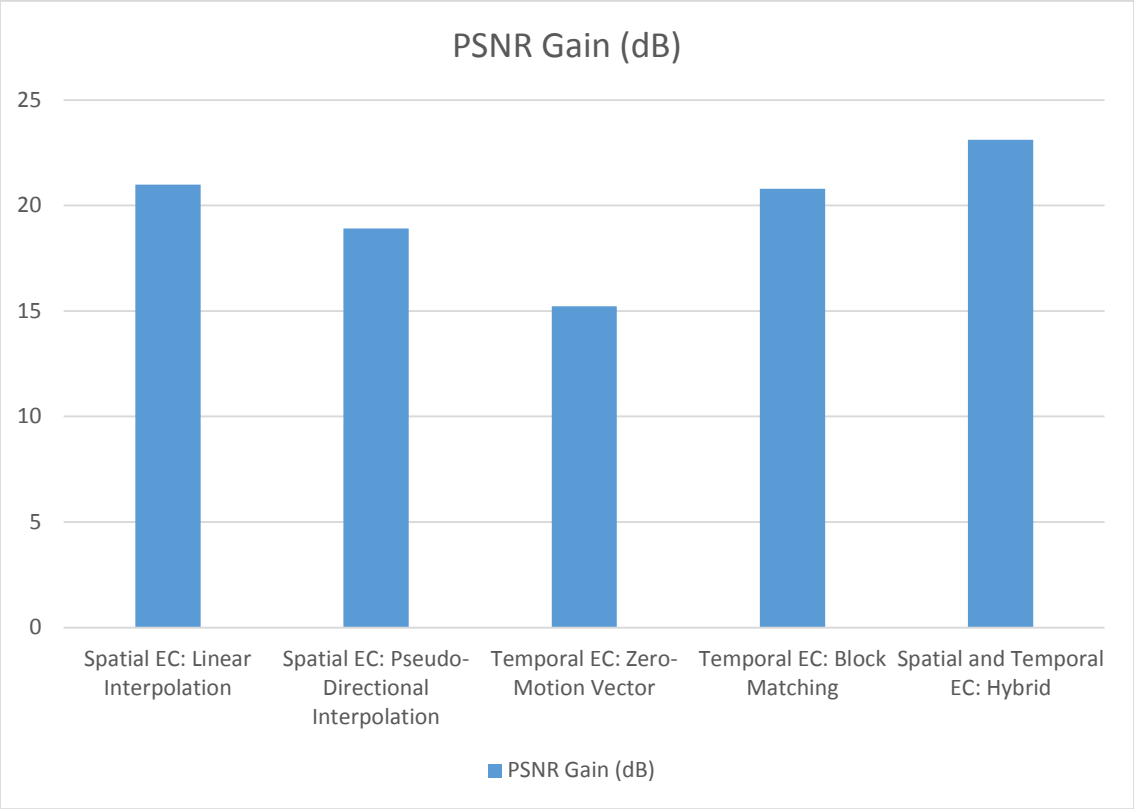
- **Spatial-Temporal Error Cocnealment**: **[HybridEC.m]**

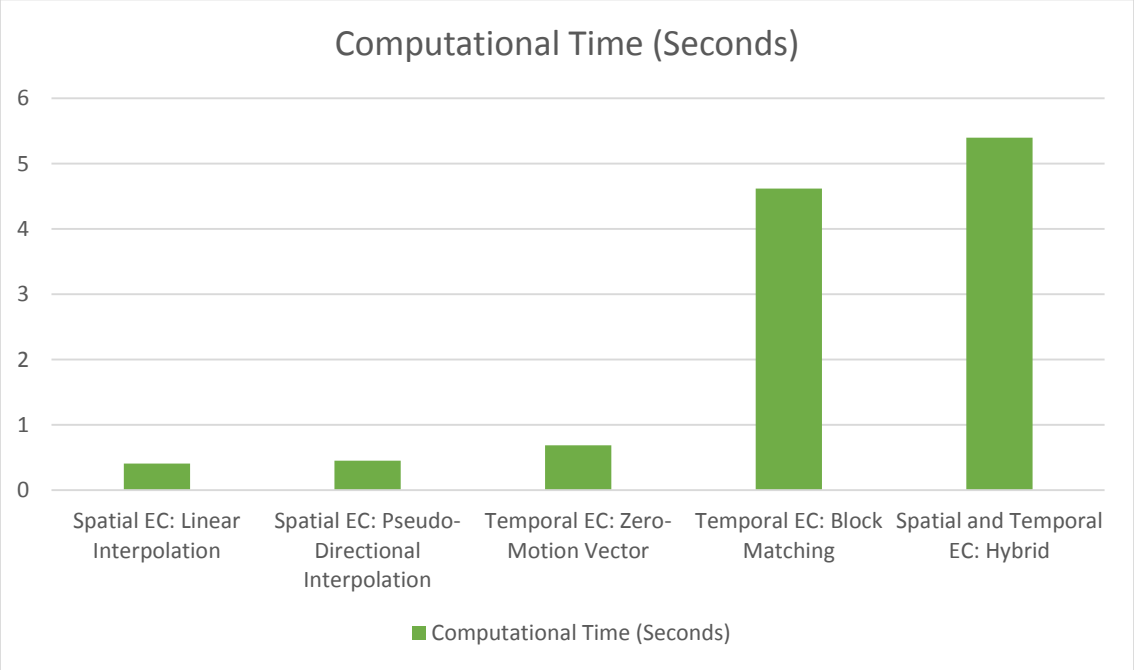


- Processing time in hybrid error concealment = 5.3976 seconds
- PSNR without error concealment = 16.7951 dB
- PSNR with error concealment = 39.9026 dB
- PSNR gain = 23,1075 dB

Analysis of the results

As expected the best performance (PSNR = 39.9026 dB and a gain of: 23,1075 dB) is achieved by the hybrid algorithm of spatial and temporal error concealment. But regarding the computational complexity, it is the method that requires the longer time to produce results (5.3976 seconds).





As a conclusion, regarding the quality of the images one will definitely choose the Hybrid method, but as the delay is highly relevant, the time processing each frame is prohibitive for this method. So taking into account both parameters, the algorithm that achieve the best ratio of quality-delay tested here so far, is the spatial error concealment using linear interpolation. This technique is the simplest and, therefore the fastest. Also the quality provided is good enough given the little complexity of the code.

REFERENCES

[1] Yao Wang, Qin-Fan Zhu *"Error Control and Concealment for Video Communication: A Review"*

[2] Paul Salama, Ness B. Shro, Edward J. Coyle, and Edward J. Delp *"Error Concealment Techniques for Encoded Video Streams"*

[3] Arvind Raman and Murali Babu *"A Low Complexity Error Concealment Scheme for MPEG-4 Coded Video Sequences"*

[4] Sofia Tsekeridou *"MPEG-2 Error Concealment Based on Block-Matching Principles"*

[5] Vineeth Shetty Kolkeri and K. R. Rao *"Error Concealment Techniques in H.264/AVC for Wireless Video Transmission in Mobile Networks"*

[6] Hamid Gharavi and Shaoshuai Gao *"Spatial Interpolation Algorithm for Error Concealment"*

[7] Dimitris. Agrafiotis, David R. Bull, Nishan Canagarajah *"Enhanced Spatial Error Concealment with Directional Entropy based Interpolation Switching"*

[8] Nourhan El Beheiry, Mohamed El Sharkawy, Mona Lotfy and Said Elnoubi *"Modifications to Fast and Efficient Spatial Error Concealment Technique for Block-Based Video Coding Systems"*

[9] Amit Kale and Vallabha Hampiholi *"Real Time Error Concealment in H.264 Video Decoder for Embedded Devices"*

[10] Shuiming Ye, Mourad Ouaret, Frederic Dufaux, Touradj Ebrahimi *"Hybrid Spatial And Temporal Error Concealment For Distributed Video Coding"*

[11] Maja Bystrom, Vasu Parthasarathy, and James W. Modestino *"Hybrid Error Concealment Schemes for Broadcast Video Transmission over ATM Networks"*

[12] Li-Wei Kang and Jin-Jang Leou *"A New Hybrid Error Concealment Scheme For Mpeg-2 Video Transmission"*

[13] Yan Chen, Yang Hu, Oscar C. Au, Houqiang Li, and Chang Wen Chen "*Video Error Concealment Using Spatio-Temporal Boundary Matching and Partial Differential Equation"*

[14] Wei-Ying Kung, Chang-Su Kim, and C.-C. Jay Kuo *"Spatial and Temporal Error Concealment Techniques for Video Transmission Over Noisy Channels"*

[15] Yan Chen, Oscar Au, Jiantao Zhou and Chi-Wang Ho *"Adaptively Switching Between Directional Interpolation and Region Matching For Spatial Error Concealment Based on DCT Coefficients"*

[16] Dimitris Agrafiotism David R. Bull and C. Nishan Canagarajah *"Enhanced Error Concealment With Mode Selection"*

[17] De-Fang Zhao, Sheng-Rong Gong and Shu-kui Zhang *"A Temporal-Domain Error Concealment Algorithm Effective In Motion Boundary Improvement"*

CODE

- **Linear Interpolation**: **[SpatialEC_linear.m]**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SPATIAL ERROR CONCEALMENT (linear interpolation)

time1s = cputime;

% USE THIS CODE FOR PROCESSING [.JPG] FORMAT TESTS
%fileName = 'lena.jpg';
%originalFrame = double(imread(fileName));
%[height,width] = size(originalFrame);

% USE THIS CODE FOR PROCESSING [.Y] FORMAT TESTS
fileName0 =  'foreman72.Y';
fileID0 = fopen(fileName0, 'r');
height = 288;
width = 352;
targetFrame = fread(fileID0, [width,height]);
originalFrame = rot90(targetFrame,3);
close;

% Setup block error parameters:
concentrationVertical = 4;
concentrationHorizontal = 4;

% Original frame
figure (1)
imshow(originalFrame, [0 255]);
title('Original frame')


frameLoss = originalFrame;
% Generation of errors in the frame
blockSize = 8;
% We are going to simulate block loss
% so that there are four neighboring blocks available for a lost block
yL = blockSize*concentrationVertical;
xL = blockSize*concentrationHorizontal;
% Possible positions of lost blocks
verticalLoss = yL:yL:(height-yL);
horizontalLoss = xL:xL:(width-xL);
for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
       frameLoss(verticalLoss(yi)+(1:blockSize), horizontalLoss(xi)+(1:blockSize)) = 0;
    end
end

% Frame with loss of information
figure (2)
imshow(frameLoss, [0 255]);
title('Frame with loss of information')

frameConcealed = frameLoss;

for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
        % First, we look for the block errors:
        if (frameLoss(verticalLoss(yi)+(1:blockSize),horizontalLoss(xi)+(1:blockSize)) == 0)
            o1 = verticalLoss(yi);
            o2 = horizontalLoss(xi);
            % For each pixel in the block:
            for i = 1:blockSize
               for j = 1:blockSize
                   % North
                   dn = i;
                   bn = frameConcealed(o1,o2+j);
                   % South
                   ds = blockSize+1-i;
                   bs = frameConcealed(o1+blockSize+1,o2+j);
```

```
                    % West
                    dw = j;
                    bw = frameConcealed(o1+i,o2);
                    % East
                    de = blockSize+1-j;
                    be = frameConcealed(o1+i,o2+blockSize+1);

                    d = dn + ds + dw + de;
                    frameConcealed(o1+i,o2+j)= (1/d) * ((dw*be)+(de*bw)+(ds*bn)+(dn*bs));
                end
            end

        end
    end
end

figure (3)
imshow(frameConcealed, [0 255]);
title('Frame with spatial error concealment')

% Calculate processing time:
time2s = cputime;
disp(['Processing time in spatial error concealment = ' num2str(time2s-time1s)]);

% Calculate PSNR
fe1 = originalFrame - frameLoss;
mse1 = mean(mean(fe1.^2));
PSNR1 = 10*log10(255^2/mse1);
disp(['PSNR without error concealment = ' num2str(PSNR1) ' dB'])

fe2 = originalFrame - frameConcealed;
mse2 = mean(mean(fe2.^2));
PSNR2 = 10*log10(255^2/mse2);
disp(['PSNR with error concealment = ' num2str(PSNR2) ' dB'])
```

- **Pseudo-Directional Interpolation**: **[SpatialEC.m]**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SPATIAL ERROR CONCEALMENT (pseudo-directional interpolation)

time1s = cputime;

% USE THIS CODE FOR PROCESSING [.JPG] FORMAT TESTS
%fileName = 'lena.jpg';
%originalFrame = double(imread(fileName));
%[height,width] = size(originalFrame);

% USE THIS CODE FOR PROCESSING [.Y] FORMAT TESTS
fileName0 =  'foreman72.Y';
fileID0 = fopen(fileName0, 'r');
height = 288;
width = 352;
targetFrame = fread(fileID0, [width,height]);
originalFrame = rot90(targetFrame,3);
close;


% Setup block error parameters:
concentrationVertical = 4;
concentrationHorizontal = 4;

% Original frame
figure (1)
imshow(originalFrame, [0 255]);
title('Original frame')


frameLoss = originalFrame;
```

```
% Generation of errors in the frame
blockSize = 8;
% We are going to simulate block loss
% so that there are four neighboring blocks available for a lost block
yL = blockSize*concentrationVertical;
xL = blockSize*concentrationHorizontal;
% Possible positions of lost blocks
verticalLoss = yL:yL:(height-yL);
horizontalLoss = xL:xL:(width-xL);
for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
       frameLoss(verticalLoss(yi)+(1:blockSize), horizontalLoss(xi)+(1:blockSize)) = 0;
    end
end

% Frame with loss of information
figure (2)
imshow(frameLoss, [0 255]);
title('Frame with loss of information')

frameConcealed = frameLoss;

for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
        % First, we look for the block errors:
        if (frameLoss(verticalLoss(yi)+(1:blockSize),horizontalLoss(xi)+(1:blockSize)) == 0)
            k = 0;
            o1 = verticalLoss(yi);
            o2 = horizontalLoss(xi);
            while (k <=(blockSize/2))

                    % North
                for g = (o1+k):(o1+(blockSize/2))
                    frameConcealed(g,o2+k) = (1/3)*( frameConcealed(g-1,o2+k) + frameConcealed(g,o2+k-1) +
frameConcealed(g-1,o2+k-1));
                end
                for g = (o1+blockSize-k):-1:(o1+(blockSize/2))
                    frameConcealed(g,o2+k) = (1/3)*( frameConcealed(g+1,o2+k) + frameConcealed(g,o2+k-1) +
frameConcealed(g+1,o2+k-1));
                end

                    % South
                for g = (o1+k):(o1+(blockSize/2))
                    frameConcealed(g,o2+blockSize-k) = (1/3)*( frameConcealed(g-1,o2+blockSize-k) +
frameConcealed(g,o2+blockSize-k+1) + frameConcealed(g-1,o2+blockSize-k+1));
                end
                for g = (o1+blockSize-k):-1:(o1+(blockSize/2))
                    frameConcealed(g,o2+blockSize-k) = (1/3)*( frameConcealed(g+1,o2+blockSize-k) +
frameConcealed(g,o2+blockSize-k+1) + frameConcealed(g+1,o2+blockSize-k+1));
                end

                    % West
                for g = (o2+k):(o2+(blockSize/2))
                    frameConcealed(o1+k,g) = (1/3)*( frameConcealed(o1+k,g-1) + frameConcealed(o1+k-1,g) +
frameConcealed(o1+k-1,g-1));
                end
                for g = (o2+blockSize-k):-1:(o2+(blockSize/2))
                    frameConcealed(o1+k,g) = (1/3)*( frameConcealed(o1+k,g+1) + frameConcealed(o1+k-1,g) +
frameConcealed(o1+k-1,g+1));
                end

                    % East
                for g = (o2+k):(o2+(blockSize/2))
                    frameConcealed(o1+blockSize-k,g) = (1/3)*( frameConcealed(o1+blockSize-k,g-1) +
frameConcealed(o1+blockSize-k+1,g) + frameConcealed(o1+blockSize-k+1,g-1));
                end
                for g = (o2+blockSize-k):-1:(o2+(blockSize/2))
                    frameConcealed(o1+blockSize-k,g) = (1/3)*( frameConcealed(o1+blockSize-k,g+1) +
frameConcealed(o1+blockSize-k+1,g) + frameConcealed(o1+blockSize-k+1,g+1));
                end

                k = k+1;
            end
        end
    end
```

```
end

figure (3)
imshow(frameConcealed, [0 255]);
title('Frame with spatial error concealment')

% Calculate processing time:
time2s = cputime;
disp(['Processing time in spatial error concealment = ' num2str(time2s-time1s)]);

% Calculate PSNR
fe1 = originalFrame - frameLoss;
mse1 = mean(mean(fe1.^2));
PSNR1 = 10*log10(255^2/mse1);
disp(['PSNR without error concealment = ' num2str(PSNR1) ' dB'])

fe2 = originalFrame - frameConcealed;
mse2 = mean(mean(fe2.^2));
PSNR2 = 10*log10(255^2/mse2);
disp(['PSNR with error concealment = ' num2str(PSNR2) ' dB'])
```

- **Zero-Motion Vector**: **[TemporalEC_zeromotion.m]**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TEMPORAL ERROR CONCEALMENT

time1t = cputime;

% Read frame samples:
fileName0 = 'foreman66.Y';
fileID0 = fopen(fileName0, 'r');
fileName1 = 'foreman72.Y';
fileID1 = fopen(fileName1, 'r');

% Frame size = (352x288)
height = 288;
width = 352;

% Read frames (we used as target frame a previous one):
targetFrame = fread(fileID0, [width,height]);
f1 = rot90(targetFrame,3); % We need to rotate the image (3x90degrees counterclockwise)
anchorFrame = fread(fileID1, [width,height]);
f2 = rot90(anchorFrame,3);
close;
close;

% Display frames:
figure (1)
imshow(uint8(f1))
title('Previous Frame')
figure (2)
imshow(uint8(f2))
title('Next Frame without block loss')

% Setup block error parameters:
concentrationVertical = 4;
concentrationHorizontal = 4;

frameLoss = f2;
% Generation of errors in the frame
blockSize = 8;
% We are going to simulate block loss
% so that there are four neighboring blocks available for a lost block
yL = blockSize*concentrationVertical;
xL = blockSize*concentrationHorizontal;
% Possible positions of lost blocks
verticalLoss = yL:yL:(height-yL);
horizontalLoss = xL:xL:(width-xL);
```

```
for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
       frameLoss(verticalLoss(yi)+(1:blockSize), horizontalLoss(xi)+(1:blockSize)) = 0;
    end
end

% Frame with loss of information
figure (3)
imshow(frameLoss, [0 255]);
title('Next Frame with block loss')

[Hloss, Wloss] = size(frameLoss);
frameConcealed = frameLoss;

for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
        % Look for the block errors:
        if (frameLoss(verticalLoss(yi)+(1:blockSize),horizontalLoss(xi)+(1:blockSize)) == 0)
            % For each lost block
            % Look for a matching block based on the neighboors
            i = verticalLoss(yi);
            j = horizontalLoss(xi);
            % Copy the block of the previous frame:
            frameConcealed(i+(1:blockSize),j+(1:blockSize)) = f1(i+(1:blockSize),j+(1:blockSize));
        end
    end
end

% Display
figure (4)
imshow(frameConcealed, [0 255]);
title('Frame with temporal error concealment')

% Calculate processing time:
time2t = cputime;
disp(['Processing time in temporal error concealment = ' num2str(time2t-time1t)]);

% Calculate PSNR
fe1 = f2 - frameLoss;
mse1 = mean(mean(fe1.^2));
PSNR1 = 10*log10(255^2/mse1);
disp(['PSNR without error concealment = ' num2str(PSNR1) ' dB'])

fe2 = f2 - frameConcealed;
mse2 = mean(mean(fe2.^2));
PSNR2 = 10*log10(255^2/mse2);
disp(['PSNR with error concealment = ' num2str(PSNR2) ' dB'])
```

- **Block-Matching**: **[TemporalEC.m]**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TEMPORAL ERROR CONCEALMENT

time1t = cputime;

% Read frame samples:
fileName0 = 'foreman66.Y';
fileID0 = fopen(fileName0, 'r');
fileName1 = 'foreman72.Y';
fileID1 = fopen(fileName1, 'r');

% Frame size = (352x288)
height = 288;
width = 352;

% Read frames (we used as target frame a previous one):
targetFrame = fread(fileID0, [width,height]);
```

```
f1 = rot90(targetFrame,3); % We need to rotate the image (3x90degrees counterclockwise)
anchorFrame = fread(fileID1, [width,height]);
f2 = rot90(anchorFrame,3);
close;
close;

% Display frames:
figure (1)
imshow(uint8(f1))
title('Previous Frame')
figure (2)
imshow(uint8(f2))
title('Next Frame without block loss')

% Setup block error parameters:
concentrationVertical = 4;
concentrationHorizontal = 4;

frameLoss = f2;
% Generation of errors in the frame
blockSize = 8;
% We are going to simulate block loss
% so that there are four neighboring blocks available for a lost block
yL = blockSize*concentrationVertical;
xL = blockSize*concentrationHorizontal;
% Possible positions of lost blocks
verticalLoss = yL:yL:(height-yL);
horizontalLoss = xL:xL:(width-xL);
for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
       frameLoss(verticalLoss(yi)+(1:blockSize), horizontalLoss(xi)+(1:blockSize)) = 0;
    end
end

% Frame with loss of information
figure (3)
imshow(frameLoss, [0 255]);
title('Next Frame with block loss')

[Hloss, Wloss] = size(frameLoss);
frameConcealed = frameLoss;

for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
        % Look for the block errors:
        if (frameLoss(verticalLoss(yi)+(1:blockSize),horizontalLoss(xi)+(1:blockSize)) == 0)
            % For each lost block
            % Look for a matching block based on the neighboors
            i = verticalLoss(yi);
            j = horizontalLoss(xi);
            range = 14;
            N =  blockSize;
            k = 10000;
            % For a defined search range we look for the matching
            % neighboors in the previous frame
            % (similar to the EBM algorithm)
            for m = -range:1:range
                for n = -range:1:range
                    % North
                    if ((i+N+m)>0 && (i+N+m+N-1)<=Hloss && (j+n)>0 && (j+n+N-1)<=Wloss)
                        a = sum(sum(abs(frameLoss(i+(1:blockSize)+N,j+(1:blockSize))-
f1(i+m+N+(1:blockSize),j+n+(1:blockSize)))));
                        if (a < k)
                            k = a;
                            dy = m;
                            dx = n;
                        end
                    end

                    % South
                    if ((i-N+m)>0 && (i-N+m+N-1)<=Hloss && (j+n)>0 && (j+n+N-1)<=Wloss)
                        a = sum(sum(abs(frameLoss(i+(1:blockSize)-N,j+(1:blockSize))-f1(i+m-
N+(1:blockSize),j+n+(1:blockSize)))));
                        if (a < k)
                            k = a;
```

```
                            dy = m;
                            dx = n;
                        end
                    end

                    % East
                    if ((i+m)>0 && (i+m+N-1)<=Hloss && (j+n+N)>0 && (j+n+N+N-1)<=Wloss)
                        a = sum(sum(abs(frameLoss(i+(1:blockSize),j+N+(1:blockSize))-
f1(i+m+(1:blockSize),j+n+N+(1:blockSize)))));
                        if (a < k)
                            k = a;
                            dy = m;
                            dx = n;
                        end
                    end

                    % West
                    if ((i+m)>0 && (i+m+N-1)<=Hloss && (j+n-N)>0 && (j+n-N+N-1)<=Wloss)
                        a = sum(sum(abs(frameLoss(i+(1:blockSize),j-N+(1:blockSize))-
f1(i+m+(1:blockSize),j+n-N+(1:blockSize)))));
                        if (a < k)
                            k = a;
                            dy = m;
                            dx = n;
                        end
                    end
                end
            end
            % Copy the best option in the lost block
            frameConcealed(i+(1:blockSize),j+(1:blockSize)) = f1(i+dy+(1:blockSize),j+dx+(1:blockSize));
        end
    end
end

% Display
figure (4)
imshow(frameConcealed, [0 255]);
title('Frame with temporal error concealment')

% Calculate processing time:
time2t = cputime;
disp(['Processing time in temporal error concealment = ' num2str(time2t-time1t)]);

% Calculate PSNR
fe1 = f2 - frameLoss;
mse1 = mean(mean(fe1.^2));
PSNR1 = 10*log10(255^2/mse1);
disp(['PSNR without error concealment = ' num2str(PSNR1) ' dB'])

fe2 = f2 - frameConcealed;
mse2 = mean(mean(fe2.^2));
PSNR2 = 10*log10(255^2/mse2);
disp(['PSNR with error concealment = ' num2str(PSNR2) ' dB'])
```

- **Spatial-Temporal Error Cocnealment**: **[HybridEC.m]**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HYBRID (SPATIAL AND TEMPORAL) ERROR CONCEALMENT

time1h = cputime;

% Read frame samples:
fileName0 = 'foreman66.Y';
fileID0 = fopen(fileName0, 'r');
fileName1 = 'foreman72.Y';
```

```
fileID1 = fopen(fileName1, 'r');

% Frame size = (352x288)
height = 288;
width = 352;

% Read frames (we used as target frame a previous one):
targetFrame = fread(fileID0, [width,height]);
f1 = rot90(targetFrame,3); % We need to rotate the image (3x90degrees counterclockwise)
anchorFrame = fread(fileID1, [width,height]);
f2 = rot90(anchorFrame,3);
close;
close;

% Display frames:
figure (1)
imshow(uint8(f1))
title('Previous Frame')
figure (2)
imshow(uint8(f2))
title('Next Frame without block loss')

% Setup block error parameters:
concentrationVertical = 4;
concentrationHorizontal = 4;

frameLoss = f2;
% Generation of errors in the frame
blockSize = 8;
% We are going to simulate block loss
% so that there are four neighboring blocks available for a lost block
yL = blockSize*concentrationVertical;
xL = blockSize*concentrationHorizontal;
% Possible positions of lost blocks
verticalLoss = yL:yL:(height-yL);
horizontalLoss = xL:xL:(width-xL);
for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
       frameLoss(verticalLoss(yi)+(1:blockSize), horizontalLoss(xi)+(1:blockSize)) = 0;
    end
end

% Frame with loss of information
figure (3)
imshow(frameLoss, [0 255]);
title('Next Frame with block loss')

[Hloss, Wloss] = size(frameLoss);
frameConcealed = frameLoss;
temporalConcealed = frameLoss;
spatialConcealed = frameLoss;

for yi = 1:length(verticalLoss)
    for xi = 1:length(horizontalLoss)
        % Look for the block errors:
        if (frameLoss(verticalLoss(yi)+(1:blockSize),horizontalLoss(xi)+(1:blockSize)) == 0)
            % For each lost block
            % Look for a matching block based on the neighboors
            i = verticalLoss(yi);
            j = horizontalLoss(xi);
            range = 14;
            N =  blockSize;
            k = 10000;
            % For a defined search range we look for the matching
            % neighboors in the previous frame
            % (similar to the EBM algorithm)
            for m = -range:1:range
                for n = -range:1:range
                    % North
                    if ((i+N+m)>0 && (i+N+m+N-1)<=Hloss && (j+n)>0 && (j+n+N-1)<=Wloss)
                        a = sum(sum(abs(frameLoss(i+(1:blockSize)+N,j+(1:blockSize))-
f1(i+m+N+(1:blockSize),j+n+(1:blockSize)))));
                        if (a < k)
                            k = a;
                            dy = m;
```

```
                            dx = n;
                        end
                    end

                    % South
                    if ((i-N+m)>0 && (i-N+m+N-1)<=Hloss && (j+n)>0 && (j+n+N-1)<=Wloss)
                        a = sum(sum(abs(frameLoss(i+(1:blockSize)-N,j+(1:blockSize))-f1(i+m-
N+(1:blockSize),j+n+(1:blockSize)))));
                        if (a < k)
                            k = a;
                            dy = m;
                            dx = n;
                        end
                    end

                    % East
                    if ((i+m)>0 && (i+m+N-1)<=Hloss && (j+n+N)>0 && (j+n+N+N-1)<=Wloss)
                        a = sum(sum(abs(frameLoss(i+(1:blockSize),j+N+(1:blockSize))-
f1(i+m+(1:blockSize),j+n+N+(1:blockSize)))));
                        if (a < k)
                            k = a;
                            dy = m;
                            dx = n;
                        end
                    end

                    % West
                    if ((i+m)>0 && (i+m+N-1)<=Hloss && (j+n-N)>0 && (j+n-N+N-1)<=Wloss)
                        a = sum(sum(abs(frameLoss(i+(1:blockSize),j-N+(1:blockSize))-
f1(i+m+(1:blockSize),j+n-N+(1:blockSize)))));
                        if (a < k)
                            k = a;
                            dy = m;
                            dx = n;
                        end
                    end
                end
            end
            % Copy the best option in the lost block
            temporalConcealed(i+(1:blockSize),j+(1:blockSize)) = f1(i+dy+(1:blockSize),j+dx+(1:blockSize));


            o1 = verticalLoss(yi);
            o2 = horizontalLoss(xi);
            % For each pixel in the block:
            for r = 1:blockSize
               for s = 1:blockSize
                    % North
                    dn = r;
                    bn = frameLoss(o1,o2+s);
                    % South
                    ds = blockSize+1-r;
                    bs = frameLoss(o1+blockSize+1,o2+s);
                    % West
                    dw = s;
                    bw = frameLoss(o1+r,o2);
                    % East
                    de = blockSize+1-s;
                    be = frameLoss(o1+r,o2+blockSize+1);

                    d = dn + ds + dw + de;
                    spatialConcealed(o1+r,o2+s)= (1/d) * ((dw*be)+(de*bw)+(ds*bn)+(dn*bs));
               end
            end


            frameConcealed(i+(1:blockSize),j+(1:blockSize)) =
0.6*spatialConcealed(i+(1:blockSize),j+(1:blockSize)) +
0.4*temporalConcealed(i+(1:blockSize),j+(1:blockSize));
        end
    end
end

% Display
```

```
figure (4)
imshow(frameConcealed, [0 255]);
title('Frame with temporal error concealment')

% Calculate processing time:
time2h = cputime;
disp(['Processing time in hybrid error concealment = ' num2str(time2h-time1h)]);

% Calculate PSNR
fe1 = f2 - frameLoss;
mse1 = mean(mean(fe1.^2));
PSNR1 = 10*log10(255^2/mse1);
disp(['PSNR without error concealment = ' num2str(PSNR1) ' dB'])

fe2 = f2 - frameConcealed;
mse2 = mean(mean(fe2.^2));
PSNR2 = 10*log10(255^2/mse2);
disp(['PSNR with error concealment = ' num2str(PSNR2) ' dB'])
```