

```
%  
% Function: MC  
% -----  
% Usage: MC_frame=MC(previous_frame,MV,Bize)  
%  
% This function creates the motion-compensated(MC) prediction of the  
% current frame by using the motion vector field, MVx and MVy, and the  
% previous frame.  
  
function MC_frame = MC(Y1, MV, Bsize)  
  
[row, col] = size(Y1);  
MC_frame = zeros([row, col]);  
  
for i=1:Bsize:row  
    for j=1:Bsize:col  
  
        mv=MV(ceil(i/Bsize), ceil(j/Bsize),:);  
  
        MC_frame(i:i+Bsize-1, j:j+Bsize-1)...  
            =Y1(i-mv(1):i-mv(1)+Bsize-1, j-mv(2):j-mv(2)+Bsize-1);  
    end  
end  
end
```

```
%
% Function: ME
% -----
% Usage: MV = ME(Bsize,Hrange,Vrange,previous_frame,current_frame)
%
% This function find a motion vector field for an entire frame by full search.
%

function MV = ME(Bsize,Hrange,Vrange,Y1,Y2)

[row col] = size(Y2);
MV = zeros([row/Bsize col/Bsize 2]);

for i=1:Bsize:row
    for j=1:Bsize:col
        % for a fixed block in the current frame,
        blk2 = Y2(i:i+Bsize-1,j:j+Bsize-1);
        min_error = inf;
        mv = [0 0];

        % Find the best match in the previous frame
        % within search window by full search

        min_y = max(i-Vrange,1);
        max_y = min(i+Vrange,row-Bsize+1);
        min_x = max(j-Hrange,1);
        max_x = min(j+Hrange,col-Bsize+1);

        for k=min_y:max_y
            for l=min_x:max_x
                % a block in the previous frame
                blk1=Y1(k:k+Bsize-1,l:l+Bsize-1);

                % MSE has mse between each blk1 and blk2
                MSE = mean(mean(abs(blk1-blk2).^2));
                % Find the index which has the MMSE
                if (MSE < min_error)
                    mv = [i-k, j-l];
                    min_error = MSE;
                end
            end
        end

        % Represent the block as a motion vector
        MV(ceil(i/Bsize),ceil(j/Bsize),:) = mv;
    end
end
end
```

```
%  
% Function: decodeBL  
% -----  
% This function decodes BL into the low resolution video seq.  
%  
function f_star = decodeBL(BL,BLMV,intra)  
  
BL_Bsize = 8;  
Fnum=size(BL,3);  
  
for j=0:intra:Fnum-1  
  
    k=1+j;  
    temp = idct2(BL(:,:,k));  
    temp(find(temp<0))=0; temp(find(temp>255))=255;  
    f_star(:,:,k) = temp;  
  
    for i=2:intra  
        i=i+j;  
        f_pred = MC(f_star(:,:,i-1),BLMV(:,:,:,i),BL_Bsize);  
        temp = f_pred + idct2(BL(:,:,i));  
        temp(find(temp<0))=0; temp(find(temp>255))=255;  
        f_star(:,:,i) = temp;  
    end  
  
end  
  
end
```

```
%  
% Function: decodeEL  
% -----  
% decode EL. If the mode is intra(0), it uses the interpolated  
% image from the BL. And if the mode is inter(1), the motion  
% predicted frame from the previous frame with motion vectors.  
  
function F_star = decodeEL(EL,ELMV,mode,f_star)  
  
EL_Bsize = 16;  
Fnum=size(EL,3);  
  
for i=1:Fnum  
    if mode(i) == 0  
        F_hat = up(f_star(:,:,i+1)/2);  
        temp = F_hat + myIDCT(EL(:,:,i));  
        temp(find(temp<0))=0; temp(find(temp>255))=255;  
        F_star(:,:,i) = temp;  
    else  
        F_pred = MC(F_star(:,:,i-1),ELMV(:,:,i),EL_Bsize);  
        temp = F_pred + myIDCT(EL(:,:,i));  
        temp(find(temp<0))=0; temp(find(temp>255))=255;  
        F_star(:,:,i) = temp;  
    end  
end
```

```
%
% Function: display.m
% -----
% 1. play recorded movie frames.
%    low resolution first and then high resolution.
% 2. make the PSNR graphs
% 3. display reconstructed frames
%
fsize=25;
for i=1:fsize
LR(i) = im2frame(f_star(:,:,i),gray(256));
end

Fsize=50;
for i=1:Fsize
HR(i) = im2frame(F_star(:,:,i),gray(256));
end

movie(LR,2,15)
pause
movie(HR,2,30)

load BL_PSNR_noerror
PSNR1=PSNR;
load BL_PSNR_previous5
PSNR2=PSNR;
load BL_PSNR_avg5
PSNR3=PSNR;
load BL_PSNR_estEL5
PSNR4=PSNR;

subplot(211)
i=0:2:50;
plot(i,PSNR1(1:25),'--',i,PSNR4(1:25),'.-',...
     i,PSNR3(1:25),'-.',i,PSNR2(1:25),'-');
xlabel('frame number');
ylabel('PSNR (dB)');
title('(a) Base Layer')
legend('Error-Free Channel','Estimation from EL','Neighbor Median',...
      'Previous Frame',4)

load EL_PSNR_noerror
PSNR1=PSNR;
load EL_PSNR_previous5
PSNR2=PSNR;
load EL_PSNR_avg5
PSNR3=PSNR;
load EL_PSNR_estEL5
PSNR4=PSNR;

subplot(212)
i=0:49;
plot(i,PSNR1,'--',i,PSNR4,'.-',i,PSNR3,'-.',i,PSNR2,'-');
xlabel('frame number');
ylabel('PSNR (dB)');
title('(b) Enhancement Layer')
legend('Error-Free Channel','Estimation from EL','Neighbor Median',...
      'Previous Frame',4)

load encBL_30_75 f_star
f1 = f_star(:,:,11);
load decBL_previous5
f2 = f_star(:,:,11);
load decBL_avg5
```

```
f3 = f_star(:,:,11);  
load decBL_estEL5  
f4 = f_star(:,:,11);
```

```
imshow(f1,[0,255]);xlabel('(a)')  
imshow(f2,[0,255]);xlabel('(b)')  
imshow(f3,[0,255]);xlabel('(c)')  
imshow(f4,[0,255]);xlabel('(d)')
```

```
%  
% Function: down  
% -----  
% Usage: qsif = down(sif)  
%  
% This function get a QCIF resolution layer from a CIF resolution  
% later by downsampling by a factor of 2.  
%  
  
function qsif = down(sif)  
  
filter = [1/2 1/2];  
  
[row col] = size(sif);  
temp = conv2(filter, filter',sif);  
  
qsif = temp(2:2:row,2:2:col);
```

```
%  
% Function: encodeBL  
% -----  
% This function encodes the qsif sequence and returns  
% BL: quantized DCT coefficients of base layer  
% f_star: reconstructed base layer without channel error,  
%       will be used when encoding EL in intra mode  
%  
function [BL,f_star] = encodeBL(qbus,BLMV,Bstep,intra)  
  
BL_Bsize = 8;  
Fnum=size(qbus,3);  
  
for j=0:intra:Fnum-1  
    k=1+j;  
    f = qbus(:, :, k);  
    BL(:, :, k) = quant(dct2(f),Bstep);  
    temp = idct2(BL(:, :, k));  
    temp(find(temp<0))=0; temp(find(temp>255))=255;  
    f_star(:, :, k) = temp;  
  
    for i=2:intra  
        i=i+j;  
        f = qbus(:, :, i);  
        f_pred = MC(f_star(:, :, i-1),BLMV(:, :, :, i),BL_Bsize);  
        r = f - f_pred;  
        BL(:, :, i) = quant(dct2(r),Bstep);  
        temp = f_pred + idct2(BL(:, :, i));  
        temp(find(temp<0))=0; temp(find(temp>255))=255;  
        f_star(:, :, i) = temp;  
    end  
  
end  
  
end
```

```
function [EL, mode] = encodeEL(bus,f_star,ELMV);

EL_Bsize = 16;
Fnum=size(bus,3);

% encode enhancement layer(EL)
% intra: predict from the base layer
% inter: predict from the enhancement layer

k=1;
F=bus(:,:,k);
F_hat = up(f_star(:,:,k));
R = F - F_hat;
EL(:,:,k) = myQuant(myDCT(R));
temp = F_hat + myIDCT(EL(:,:,k));
temp(find(temp<0))=0; temp(find(temp>255))=255;
F_star(:,:,k) = temp;
mode(k) = 0;

for i=2:2:Fnum-1
    F = bus(:,:,i);
    F_pred = MC(F_star(:,:,i-1),ELMV(:,:,i),EL_Bsize);
    R = F - F_pred;
    EL(:,:,i) = myQuant(myDCT(R));
    temp = F_pred + myIDCT(EL(:,:,i));
    temp(find(temp<0))=0; temp(find(temp>255))=255;
    F_star(:,:,i) = temp;
    mode(i) = 1;

    F = bus(:,:,i+1);

    F_pred = MC(F_star(:,:,i),ELMV(:,:,i+1),EL_Bsize);
    R1 = F - F_pred;
    EL1 = myQuant(myDCT(R1));
    temp = F_pred + myIDCT(EL1);
    temp(find(temp<0))=0; temp(find(temp>255))=255;
    F_star1 = temp;
    MSE1 = mean(mean(abs(F-F_star1).^2));

    F_hat = up(f_star(:,:,i+2)/2));
    R2 = F - F_hat;
    EL2 = myQuant(myDCT(R2));
    temp = F_hat + myIDCT(EL2);
    temp(find(temp<0))=0; temp(find(temp>255))=255;
    F_star2 = temp;
    MSE2 = mean(mean(abs(F-F_star2).^2));

    if MSE1<MSE2
        EL(:,:,i+1) = EL1;
        F_star(:,:,i+1) = F_star1;
        mode(i+1) = 1;
    else
        EL(:,:,i+1) = EL2;
        F_star(:,:,i+1) = F_star2;
        mode(i+1) = 0;
    end
end

end

l=Fnum;
F = bus(:,:,l);
F_pred = MC(F_star(:,:,l-1),ELMV(:,:,l),EL_Bsize);
R = F - F_pred;
EL(:,:,l) = myQuant(myDCT(R));
```

```
temp = F_pred + myIDCT(EL(:,:,1));  
temp(find(temp<0))=0; temp(find(temp>255))=255;  
F_star(:,:,1) = temp;  
mode(1) = 1;
```

```
%  
% Function: estMV  
% -----  
% Estimate the lost motion vectors by using four surrounding vectors.  
%  
function MV_est = estMV(MV,ErrorMask,n)  
  
MV_est = MV;  
mv = MV(:,:, :,n);  
  
k=1:2;  
  for i=2:14  
    for j=2:21  
      if ErrorMask(i,j) == 0  
        mv(i,j,k) = round(median(...  
          [mv(i,j-1,k) mv(i,j+1,k) mv(i-1,j,k) mv(i+1,j,k)]));  
%        mv(i,j,k) = round(mean(...  
%          mv(i,j-1,k)+mv(i,j+1,k)+mv(i-1,j,k)+mv(i+1,j,k)));  
        end  
      end  
    end  
  end  
  
MV_est(:,:, :,n) = mv;
```

```
% if index = 0, weighted motion vectors

function MV_estEL = estMVfromEL(MV,ErrorMask,n,ELMV);

Bsize = 16;
row = 240;
col = 352;

MV_estEL = MV;

for i=1:15
    for j=1:22
        if ErrorMask(i,j) == 0
            mv1(1) = ELMV(i,j,1,2*n-1);
            mv1(2) = ELMV(i,j,2,2*n-1);

            % map has one in the block pointed by mv, zero otherwise
            map = zeros([row,col]);
            map(Bsize*(i-1)+1-mv1(1):Bsize*i-mv1(1), ...
                Bsize*(j-1)+1-mv1(2):Bsize*j-mv1(2)) = 1;

            % calculate weight for any block overlapped by map
            for k=1:Bsize:row
                for l=1:Bsize:col
                    weight(ceil(k/Bsize), ceil(l/Bsize))= ...
                        sum(sum(map(k:k+Bsize-1, l:l+Bsize-1)))/16^2;
                end
            end

            mv2=zeros(1,2);

            % weighted sum of 4 motion vectors
            for k=1:15
                for l=1:22
                    if weight(k,l) ~= 0
                        mv2(1) = mv2(1) + weight(k,l)*ELMV(k,l,1,2*n-2);
                        mv2(2) = mv2(2) + weight(k,l)*ELMV(k,l,2,2*n-2);
                    end
                end
            end
            mv = round((mv1 + mv2)/2);
            MV_estEL(i,j,1,n) = mv(1);
            MV_estEL(i,j,2,n) = mv(2);
        end
    end
end
```

```
%
% File: generate.m
% -----
% This file generates input sequence for both layer and calculate
% motion vectors for the motion compensation at encoder and decoder.
%
% -----
% make bus, only with luminance of bus sequence.
% -----
for i=1:150
    [y u v]=read_frame_sif('bus',i-1);
    bus(:,:,i)=y;
end
save bus2.mat bus

% -----
% make qbus, downsampled version of bus.
% -----
for i=1:75
    qbus(:,:,i) = down(bus(:,:,2*i-1));
end
save qbus.mat qbus

% -----
% store motion vectors for original images
% this motion vectors used for motion compensation
% reduce a big amount of computational time
% -----

% constants
EL_Bsize=16; BL_Bsize=8; % block size
EL_Hrange=24; BL_Hrange=12; % horizontal search range
EL_Vrange=16; BL_Vrange=8; % vertical search range

% EL
for i=2:150
    ELMV(:,:,,i)=ME(EL_Bsize,EL_Hrange,EL_Vrange,bus(:,:,i-1),bus(:,:,i));
end
save ELMV.mat ELMV

% BL lowers the frame rate by dropping frames
% need only a half motion vectors
for i=3:2:150
    BLMV(:,:,,i)=ME(BL_Bsize,BL_Hrange,BL_Vrange,qbus(:,:,i-2),qbus(:,:,i));
end
save BLMV2.mat BLMV
```

```
%
% File: main.m
% -----
% This is a main program to simulate the trasmission system
% including encoder, channel, decoder.
%
% -----
% encode BL:
% number of frames should be an integer multiple of intra,
% which is the frequency of intra mode
% -----
load qbus.mat qbus
load BLMV.mat BLMV

Bstep=30; % quantization step for BL
intra=75;
[BL f_star] = encodeBL(qbus,BLMV,Bstep,intra);
save encBL_30_75 BL f_star
clear BL

i=1:75;
PSNR(i) = ...
10*log10(255^2/mean(mean(abs(qbus(:, :, i)-f_star(:, :, i)).^2)));
save BL_PSNR_noerror PSNR
clear

% -----
% encode EL:
% due to the limit of memory, bus should be separated into a few
% shorter sequences before encoded
% -----
load bus1.mat
load ELMV.mat ELMV
load encBL_30_75 f_star
[EL mode] = encodeEL(bus1,f_star(:, :, 1:25), ELMV(:, :, :, 1:50));
save encEL EL mode
clear

% -----
% decode EL with error-free channel
% -----
load encEL.mat EL mode
load encBL_30_75.mat f_star
load ELMV.mat ELMV
F_star = decodeEL(EL,ELMV,mode,f_star);
save decEL_noerror F_star
clear EL; clear f_star
load bus1.mat

i=1:50;
PSNR(i)=...
10*log10(255^2/mean(mean(abs(bus1(:, :, i)-F_star(:, :, i)).^2)));
save EL_PSNR_noerror PSNR
clear

% -----
% channel error: lose 10% of motion vectors in BL
% -----
load encBL_30_75.mat BL
load BLMV2.mat BLMV

% error on BL motion vectors on frame #n
ErrorMask = rand(15,22);
ErrorMask(find(ErrorMask>0.9))=0;
```

```
ErrorMask(find(0<ErrorMask & ErrorMask<0.9))=1;
save ErrorMask ErrorMask

n=5;
BLMV_error = BLMV;
BLMV_error(:,:,1,n) = ErrorMask.*BLMV(:,:,1,n);
BLMV_error(:,:,2,n) = ErrorMask.*BLMV(:,:,2,n);

% -----
% all motion vectors in error are set to zero
% (previous frame concealment)
% -----
f_star = decodeBL(BL,BLMV_error,75);
save decBL_previous f_star
clear BL; load qbus;

i=1:75;
PSNR(i) = ...
    10*log10(255^2/mean(mean(abs(qbus(:,:,i)-f_star(:,:,i)).^2)));
save BL_PSNR_previous PSNR
clear qbus;

load encEL.mat EL mode
load ELMV
F_star = decodeEL(EL,ELMV,mode,f_star);
save decEL_previous F_star
clear EL; clear f_star;

load bus1;
i=1:50;
PSNR(i)=...
    10*log10(255^2/mean(mean(abs(bus1(:,:,i)-F_star(:,:,i)).^2)));
save EL_PSNR_previous PSNR
clear

% -----
% MV estimation by averaging of four motion vectors in neighborhood
% -----
load BLMV2.mat BLMV
load ErrorMask,mat ErrorMask

n=5;
BLMV_error = BLMV;
BLMV_error(:,:,1,n) = ErrorMask.*BLMV(:,:,1,n);
BLMV_error(:,:,2,n) = ErrorMask.*BLMV(:,:,2,n);

MV_est = estMV(BLMV_error,ErrorMask,5);
load encBL_30_75 BL
f_star = decodeBL(BL,MV_est,75);
save decBL_avg f_star
clear BL; load qbus

i=1:75;
PSNR(i) = ...
    10*log10(255^2/mean(mean(abs(qbus(:,:,i)-f_star(:,:,i)).^2)));
save BL_PSNR_avg PSNR
clear qbus

load encEL EL mode
load ELMV
F_star = decodeEL(EL,ELMV,mode,f_star);
save decEL_avg F_star
clear EL; clear f_star;
load bus1;
```

```
i=1:50;
PSNR(i)=...
    10*log10(255^2/mean(mean(abs(bus1(:,:,i)-F_star(:,:,i)).^2)));
save EL_PSNR_avg5 PSNR
clear

% -----
% MV estimation by averaging two motion vectors in EL
% -----
load ELMV.mat ELMV
load BLMV2.mat BLMV
load ErrorMask.mat ErrorMask

n=5;
BLMV_error = BLMV;
BLMV_error(:,:,1,n) = ErrorMask.*BLMV(:,:,1,n);
BLMV_error(:,:,2,n) = ErrorMask.*BLMV(:,:,2,n);

MV_estEL = estMVfromEL(BLMV_error,ErrorMask1,n,ELMV);

load encBL_30_75 BL
f_star = decodeBL(BL,MV_estEL,75);
save decBL_estEL f_star
clear BL; load qbus
i=1:75;
PSNR(i) = ...
    10*log10(255^2/mean(mean(abs(qbus(:,:,i)-f_star(:,:,i)).^2)));
save BL_PSNR_estEL PSNR
clear qbus

load encEL.mat EL mode
F_star = decodeEL(EL,ELMV,mode,f_star);
save decEL_estEL F_star
clear f_star; clear EL;
load bus1;
i=1:50;
PSNR(i)=...
    10*log10(255^2/mean(mean(abs(bus1(:,:,i)-F_star(:,:,i)).^2)));
save EL_PSNR_estEL PSNR
```

```
function out = myDCT(im)

[row col] = size(im);
out = zeros([row col]);

for i=1:8:row
    for j=1:8:col
        out(i:i+7,j:j+7) = dct2(im(i:i+7,j:j+7));
    end
end
```

```
function out = myIDCT(in)

[row col] = size(in);
out = zeros([row col]);

for i=1:8:row
    for j=1:8:col
        out(i:i+7,j:j+7) = idct2(in(i:i+7,j:j+7));
    end
end
```

```
function out = myQuant(in);  
  
out = in;  
  
num = prod(size(in));  
vec = reshape(in,num,1);  
sorted_vec = sort(abs(vec));  
Th = sorted_vec(0.9*num+1);  
  
out(find(abs(in)<Th)) = 0;
```

```
%  
% Function: up  
% -----  
% Usage: sif = up(qsif)  
%  
% This function get a SIF (352x240 pels) resolution layer from a QSIF  
% (176x120 pels) resolution layer by interpolation with a filtering  
% [1/4 3/4 3/4 1/4] in the horizontal and vertical direction.  
%  
function sif = up(qsif)  
  
[row,col] = size(qsif);  
  
i=1:row;  
j=1:col-1;  
  
temp(i,1) = 3/4*qsif(i,1);  
temp(i,col*2) = 3/4*qsif(i,col);  
  
temp(i,2*j) = 3/4*qsif(i,j) + 1/4*qsif(i,j+1);  
temp(i,2*j+1)=1/4*qsif(i,j) + 3/4*qsif(i,j+1);  
  
j=1:col*2;  
i=1:row-1;  
  
sif(1,j) = 3/4*temp(1,j);  
sif(row*2,j) = 3/4*temp(row,j);  
  
sif(2*i,j) = 3/4*temp(i,j) + 1/4*temp(i+1,j);  
sif(2*i+1,j)=1/4*temp(i,j) + 3/4*temp(i+1,j);
```