Report for EQ2330 Image and Video Processing

EQ2330 Image and Video Processing, Project 3

Zhenghao Li zhenghl@kth.se
Letao Feng letao@kth.se
Heng Zhao hengzhao@kth.se

December 20, 2023

Summary

In this project, we study three video coding algorithms and evaluate their performance. Based on the Intra-Frame Video Coder, we expand it to the Conditional Replenishment Video Coder and then study the Video Coder with Motion Compensation. The Intra-Frame Video Coder is based on DCT transform, encoding each frame in the video sequence independently of the other frames. The Conditional Replenishment Video Coder can rely on other frames to reconstruct the image and has two modes, the intra mode and the copy mode. The Video Coder with Motion Compensation introduces a new mode, inter mode, to the aforementioned two modes. The model decision is based on the minimization of the Lagrangian cost function. To evaluate their performance, we choose the bit-rate and PSNR as the metrics and then compare these rate-PSNR curves.

1 Introduction

In this project, three video coding methods are implemented: Intra-Frame Video Coder, Conditional Replenishment Video Coder, and Motion-Compensated Video Coder. Then, we encode the luminance signals of the first 50 frames of the QCIF sequences "Foreman" and "Mother-Daughter" respectively. Finally, we evaluate their performance by varying the step size of the quantizer and then measuring the rate-PSNR curve.

2 System Description

2.1 Intra-Frame Video Coder

The intra-frame coder codes each frame of the video sequence, independently of any other frames.

In this section, we first divide each frame into 16×16 blocks and apply four two-dimensional 8×8 DCT transforms for each 16×16 block. Then, we use a uniform mid-tread quantizer to implement the quantization of the coefficients. Finally, the reconstructed frame can be obtained by applying IDCT to the quantized DCT coefficients.

According to the Parseval Theorem, to calculate the block distortion, we only need to calculate the mean squared error (MSE) between the original and the quantized DCT coefficients. Then, PSNR can be calculated as follows.

$$PSNR = 10 \log_{10}(\frac{255^2}{d}) \tag{1}$$

where d is the mean squared error (MSE) between the original and the reconstructed block.

According to the Noiseless Source Coding Theorem, the required number of bits to code the one pixel is approximated as the entropy. In this case, to measure the bit-rate of the video with a 30 frames per second in kbit/s, we have

$$Bitrate = \frac{Entropy \cdot 176 \cdot 144 \cdot 30}{50 \cdot 1000} \tag{2}$$

2.2 Conditional Replenishment Video Coder

Since the Intra-Frame Video Coder has a high bit rate, we modify it to make it a Conditional Replenishment Video Coder that can rely on other frames to reconstruct the image (block).

There are two modes: the intra mode and the copy mode. The intra mode is the intra-frame coding mentioned above, and the copy mode is performed by copying the 16×16 block located at the same position in

the previous frame to the current 16×16 block.

To decide which mode to choose is based on the results of the Lagrangian cost function

$$J_n = D_n + \lambda R_n \tag{3}$$

where D_n denotes the MSE for mode n, R_n denotes the bit-rate(entropy) for mode n. In this case, $\lambda = 0.2Q^2$, where Q denotes the step size. We can use Formula 1 to calculate D_n and the entropy per pixel in one block to approximate R_n. Additionally, R_n needs 1 more bit to distinguish between the two modes.

For each 16×16 block, we choose the mode n that results in the minimum Lagrangian cost J_n.

2.3 Video Coder with Motion Compensation

In this section, a new mode, inter mode, is implemented in the conditional replenishment encoding system in addition to the aforementioned two modes. The purpose is to exploit, to a greater extent, the inter-frame relationship in the video.

The new mode consists of two parts: Motion-Compensated Prediction and Residual Coding.

In the Motion-Compensated Prediction part, a motion-compensated prediction with the displacement range of $[-10, ..., 10] \times [-10, ..., 10]$ pel is performed to locate the most similar block in the previous frame stored. The relative position of the displacement and the to-be-coded block is stored as the motion vector, which require $2 \cdot \lceil log_2 21 \rceil = 10$ bits per block in addition to 2 bits per block to indicate which coding mode is chosen.

Next, we conduct a deduction between the to-be-coded block and the prediction block searched in the last frame. To achieve a higher approximation level, the residual is coded in the same manner that a block is coded with intra mode. Hence, the required bits to store the residual can be calculated using the Formula 2.

In conclusion, for every block, $R = 12 + entropy\ of\ resiual$ bits is needed to code a block with inter mode. To compute PSNR, the distortion d is calculated as the MSE between the original and the reconstructed block. Here, we add the reconstructed residual to the prediction block to acquire the reconstructed block.

3 Results

3.1 Intra-Frame Video Coder

In Figure1 the quantizer step-size increased form right to left, which means that more pixel values are mapped to the same discrete value. As a consequence, the number of bits required for encoding are decreased. This leads to a loss of information in the image as adjacent pixels are mapped to the same value. This is shown in the figure as the PSNR gradually decreases.

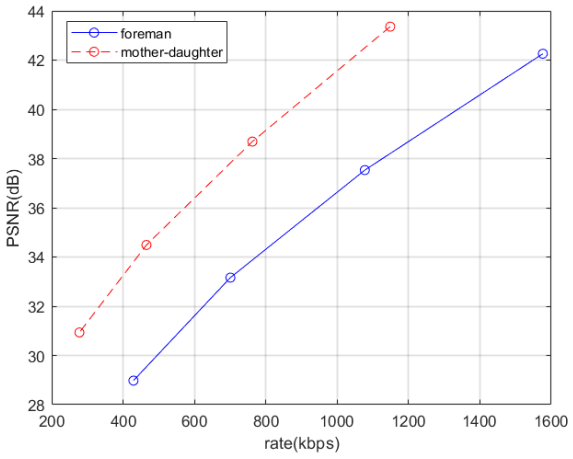
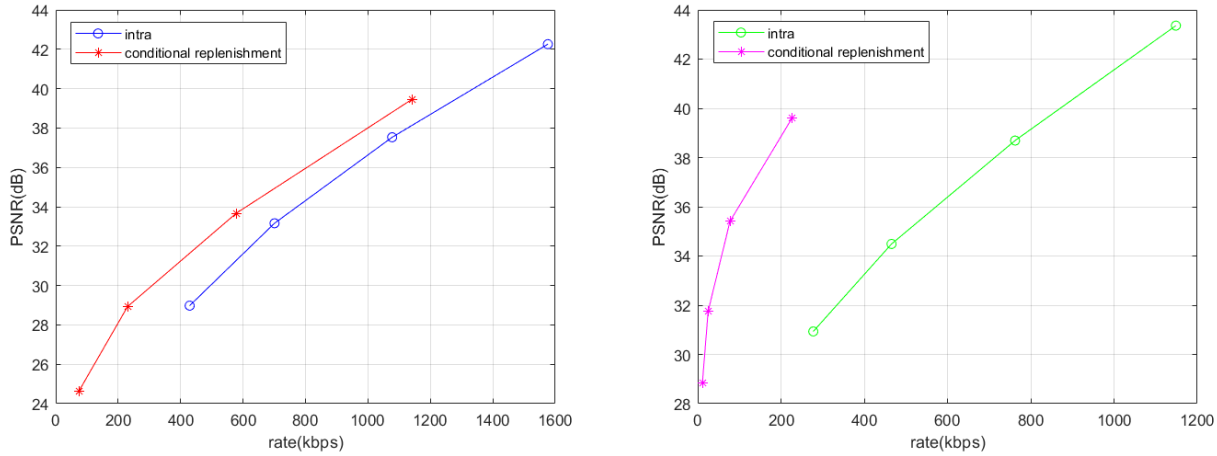


Figure 1: rate-PSNR curve of intra-frame video coder

3.2 Conditional Replenishment Video Coder

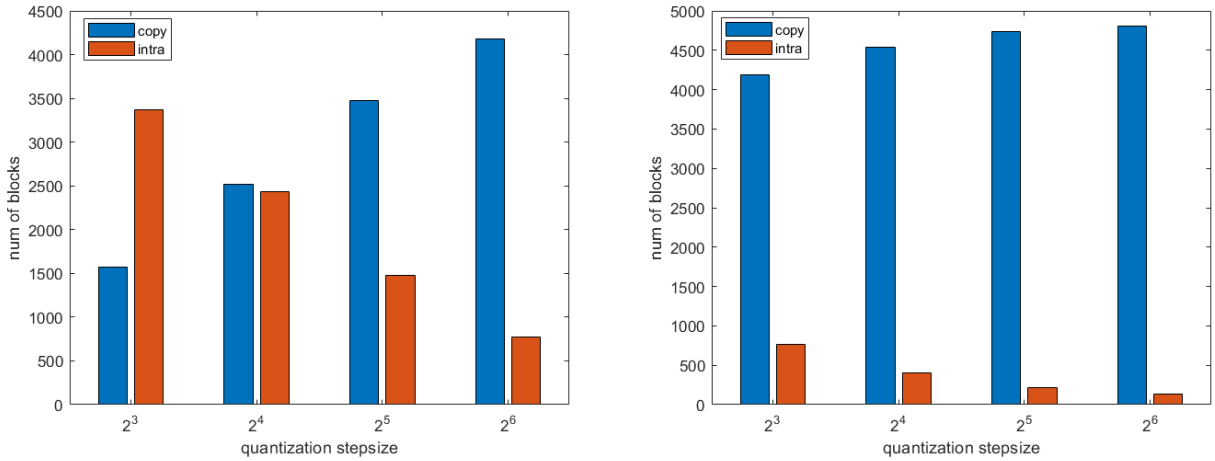
Figure2 shows the rate-PSNR curve of two videos. For “foreman” video, at the same PSNR, the bit rate required for conditional replenishment video coder can reduced by 150-200 kbps compared with intra-frame coding. For

"mother-daughter" video, the performance of conditional replenishment video coder is significantly improved. In the quantization step range of 8-64, under the same distortion levels, the code rate required for conditional replenishment video coder can be reduced by 300-600 kbps compared with intra-frame coding.



(a) Foreman (b) Mother-daughter

Figure 2: rate-PSNR curve of conditional replenishment video coder and intra-frame video coder



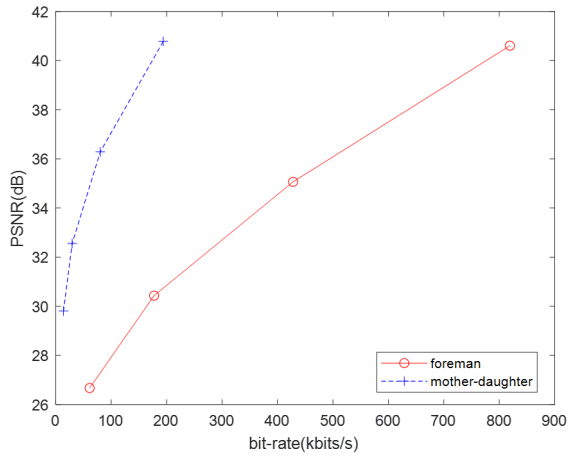
(a) Foreman (b) Mother-daughter

Figure 3: number of modes used in conditional replenishment video coder

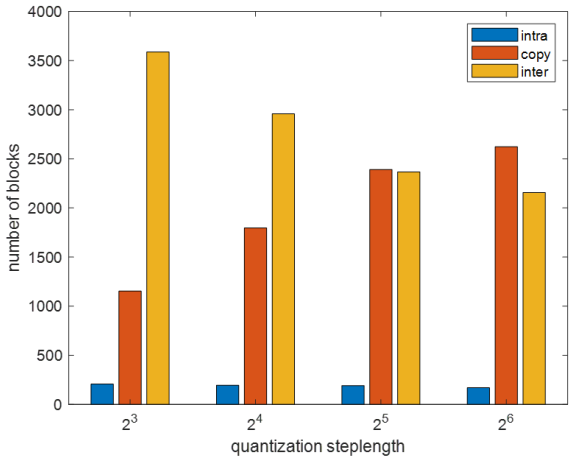
Figure 3 shows the number of blocks using intra mode and copy mode respectively in two videos. As the quantizer step-size increases, frame distortion is become more severe and the copy mode is used more. Compared with "foreman" video, more copy modes are used in the compression of "mother-daughter" video. Because the background of this video is relatively single, and there is almost no relative movement in the background between two adjacent frames of images.

3.3 Video Coder with Motion Compensation

Figure 4a shows the rate-PSNR curve of the coder with three modes coding two videos, while figure 4b shows the number of coding modes used respectively at different quantizing step-size. The figure on the left indicates an increase of roughly 6dB each one more bit in coding one pixel at high bit-rate. The number of modes used indicates a superiority of the inter mode that most blocks have chosen it to cut the Lagrangian Loss when the step size is relatively low. This may be attributed to the compromise inter mode makes between reducing the bit-rate and the distortion. However, as the step size increases, more blocks decide to use copy mode. This is reasonable, because with greater step size, the residual is coded with greater distortion that inter mode becomes



(a) rate-PSNR curve of Coder with Motion Compensation



(b) number of modes used

Figure 4: Result of Video Coder with Motion Compensation Coding Foreman

inefficient.

Figure 5 shows the performances of the aforementioned three kinds of coders when coding the video foreman. It is clear that the one with three modes has the best performance as it can save more bits when compressing the video to the same quality. To analyse more deeply, the quality of the compressed video is highest when using only intra-frame coder, then when using coder with motion compensation and at last using conditional replenishment coder. The essence to this phenomenon is that by exploiting the inter-frame relationship, motion compensation coder is able to encoding the current frame with less bits comparing to intra mode and less distortion comparing to just copying the block in the last frame.

A more sophisticated coder can be proposed based on the result above is to search for a best-fitted block in the last frame that may have some extent of rotation comparing to the to-be-coded block. What can be foreseen is that this coder will introduce a larger code rate but a better quality and that will be chosen more than other modes when the quantization step size is relatively small. However, this will increase the complexity of coders and decoders.

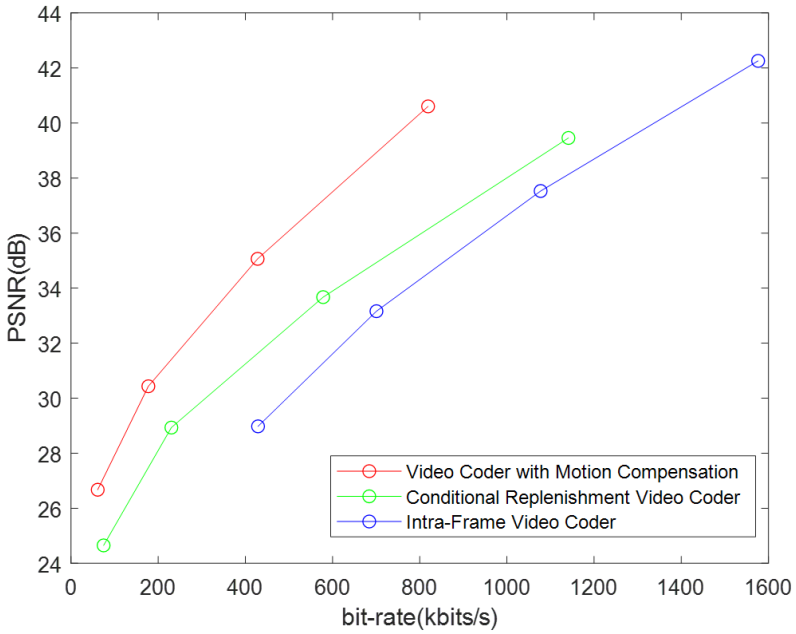


Figure 5: Comparison of three kinds of coders

4 Conclusion

In this project, three video coders were implemented to achieve video compression using intra, copy, and inter mode, exploiting both spatial and temporal redundancy. Spatial redundancy is addressed by applying a discrete cosine transform (DCT) on small video blocks, with coefficients encoded using variable length coding (VLC). This transformation is particularly effective for videos with a constant background, enabling significant compression by encoding low-frequency components with shorter codewords.

Temporal redundancy is tackled through copy mode and inter mode. Copy mode involves replicating previous blocks, resulting in substantial compression for videos with constant backgrounds—making it particularly crucial for video conference systems. Motion mode addresses motion in the video by identifying similar blocks in the previous frame and encoding only the residual changes. This mode efficiently encodes moving objects, such as heads, by capturing their position without encoding every pixel change. Motion mode avoids the costly encoding of moving parts seen in intra mode and is considered a valuable component for efficient video encoding.

For mode selection, a Lagrangian loss function was implemented, weighing the compression rate against distortion. Experimental results emphasize the significance of optimizing the weight parameter to meet the specific application's requirements, achieving the best trade-off between compression rate and distortion.

Finally, an idea of a more advanced video coder is proposed which could be developed by searching for a best-fitted block in the previous frame, even with some rotation. This may result in a higher code rate but improved quality, especially with a small quantization step size. However, it comes at the expense of increased complexity in both coders and decoders.

Appendix

Who Did What

	contribution
Letao Feng	33.3...%
Heng Zhao	33.3...%
Zhenghao Li	33.3...%

MatLab code

```
main1


%mother-daughter_qcif/mother-daughter_qcif.yuv /
% foreman_qcif\foreman_qcif.yuv
height=144;
width=176;
numfram=50;
size=16;
Y = yuv_import_y('mother-daughter_qcif/mother-daughter_qcif.yuv',[width height],numfram);

numblock=(width/16)*(height/16);
stepsize=[8,16,32,64];

f=zeros(height,width,numfram);
for i=1:50
    f(:,:,i)=Y{i,1};
end

RR=zeros(1,4);
DD=zeros(1,4);

for q=3:6
rr=zeros(height/16,width/16,numfram);
dd=zeros(height/16,width/16,numfram);
m=1:16;
```

```
for i= 1:numfram
    for row=1:height/16
        for column=1:width/16
T=dct16(f(16*(row-1)+m,16*(column-1)+m,i));
Q=midtread(T,2^q);

rr(row,column,i)=entr(Q);
dd(row,column,i)=immse(T,Q);
        end
    end
end

RR(1,q-2)=sum(rr,"all");
DD(1,q-2)=sum(dd,"all");

end
r=RR/50/1000*30*256;
P=10*log10(255*255./(DD/50/99));
plot(r,P,'go-');
xlabel('rate(kbps)');
ylabel('PSNR(dB)')
grid on



main2

height=144;
width=176;
numfram=50;
size=16;
Y = yuv_import_y('mother-daughter_qcif/mother-daughter_qcif.yuv',[width height],numfram);
% numblock=(width/16)*(height/16);
% stepsize=[8,16,32,64];

f=zeros(height,width,numfram);
for i=1:50
    f(:,:,i)=Y{i,1};
end

R=zeros(height/16,width/16,numfram);
D=zeros(height/16,width/16,numfram);
R_plot=zeros(1,4);
D_plot=zeros(1,4);
m=1:16;
num=zeros(2,4);

for q=3:6
%the first frame
r1=RATE(f(:,:,1),2^q);
d1=intra_mode_d(f(:,:,1),2^q);
R(:,:,1)=r1;
D(:,:,1)=d1;

for j=2:50
    last_frame=f(:,:,j-1);
    current_frame=f(:,:,j);

    d_copy=copy_mode_d(last_frame,current_frame,2^q);
```

```
    d_intra=intra_mode_d(current_frame,2^q);

    r_copy=zeros(height/16,width/16)+1/256;
    r_intra=RATE(current_frame,2^q);

    r_final=zeros(height/16,width/16);
    d_final=zeros(height/16,width/16);

    J_copy=Lagrangian_cost(d_copy,r_copy,2^q);
    J_intra=Lagrangian_cost(d_intra,r_intra,2^q);

    selection=J_intra-J_copy;

    for row=1:height/16
        for column=1:width/16
            if(selection(row,column)>=0)
                r_final(row,column)=r_copy(row,column);
                d_final(row,column)=d_copy(row,column);

                f_dct=dct16(last_frame(16*(row-1)+m,16*(column-1)+m));
                f_q=midtread(f_dct,2^q);
                f_idct=idct16(f_q);
                current_frame(16*(row-1)+m,16*(column-1)+m)=f_idct;
                f(:,:,j)=current_frame;

            else
                r_final(row,column)=r_intra(row,column);
                d_final(row,column)=d_intra(row,column);
            end
        end
        R(:,:,j)=r_final;
        D(:,:,j)=d_final;
    end
end
num(1,q-2)=length(find(R==0.003906250000000));
num(2,q-2)=99*50-num(1,q-2);

R_plot(1,q-2)=sum(R,"all");
D_plot(1,q-2)=sum(D,"all");

end

R_plot=R_plot/50/1000*30*256;
PSNR=10*log10(255*255./(D_plot/50/99));
plot(R_plot,PSNR,'y*-');
grid on
% figure
%
% bar(num');
% xticklabels({'2^3', '2^4', '2^5', '2^6'});
% xlabel('quantization stepsize');
% ylabel('num of blocks')
% legend('copy','intra');



main3

Y = yuv_import_y('mother-daughter_qcif/mother-daughter_qcif.yuv',[176 144],50);
```

```
M=176/16;
N=144/16;

rec_video = cell(50,1);
D = zeros(1,4);
R = zeros(1,4);

cnt_intra = zeros(1,4);
cnt_copy = zeros(1,4);
cnt_inter = zeros(1,4);

for l = 3:6
    % for the first frame, only intra mode can be chosen.
    cur_frame = Y{1};
    [d,r,rec_image] = intra_mode(cur_frame, 2^l);
    rec_video{1} = rec_image;

    D(l-2) = sum(d,"all");
    R(l-2) = sum(r,"all");
    cnt_intra(l-2) = 99;

    for frame = 2:50

        cur_frame = Y{frame};
        last_rec_frame = rec_video{frame-1};

        [d_intra,r_intra,rec_image_intra] = intra_mode(cur_frame, 2^l);
        [d_copy,r_copy,rec_image_copy] = copy_mode(cur_frame, last_rec_frame);
        [d_inter,r_inter,rec_image_inter] = inter_mode(cur_frame, last_rec_frame, 2^l);
        [d,r,rec_image,choice] = mode_choice(d_intra,r_intra,rec_image_intra,...
    d_copy,r_copy,rec_image_copy,...
    d_inter,r_inter,rec_image_inter,...
    2^l);
        rec_video{frame} = rec_image;
        D(l-2) = D(l-2) + sum(d,"all");
        R(l-2) = R(l-2) + sum(r,"all");
        cnt_intra(l-2) = cnt_intra(l-2) + length(find(choice==1));
        cnt_copy(l-2) = cnt_copy(l-2) + length(find(choice==2));
        cnt_inter(l-2) = cnt_inter(l-2) + length(find(choice==3));
    end
end

R = R/50/99*144*176*30/1000;
D = D/50/99;
PSNR = 10*log10(255*255./D);
plot(R,PSNR,'ro-')



function [d,r,rec_image,choice] = mode_choice(d_intra,r_intra,rec_image_intra,...
    d_copy,r_copy,rec_image_copy,...
    d_inter,r_inter,rec_image_inter,...
    steplen)

M = 176/16;
N = 144/16;
```

```
lambda = 0.2 * steplen^2;

d = zeros(9,11);
r = zeros(9,11);
rec_image = zeros(144,176);

J = zeros(9,11,3);
D = zeros(9,11,3);
R = zeros(9,11,3);
REC = zeros(144,176,3);

J(:,:,1) = d_intra + lambda * r_intra;
J(:,:,2) = d_copy + lambda * r_copy;
J(:,:,3) = d_inter + lambda * r_inter;

D(:,:,1) = d_intra;
D(:,:,2) = d_copy;
D(:,:,3) = d_inter;

R(:,:,1) = r_intra;
R(:,:,2) = r_copy;
R(:,:,3) = r_inter;

REC(:,:,1) = rec_image_intra;
REC(:,:,2) = rec_image_copy;
REC(:,:,3) = rec_image_inter;


[~, choice] = min(J,[],3);
for i = 1:M
    for j = 1:N
        d(j,i) = D(j,i,choice(j,i));
        r(j,i) = R(j,i,choice(j,i));
        rec_image((j - 1) * 16 + (1:16),(i - 1) * 16 + (1:16)) = REC((j - 1) * 16 + (1:16),(i - 1) * 16
    end
end

end


function [d,r,rec_image] = intra_mode(image,steplen)

M = 176/16;
N = 144/16;

d = zeros(N,M);
r = zeros(N,M);

dct_co = blkproc(image, [8 8], @dct2);
dct_co_qtz = midtread(dct_co,steplen);
rec_image = blkproc(dct_co_qtz, [8 8], @idct2);

for i = 1:M
    for j = 1:N
        d(j,i) = immse(rec_image((j - 1) * 16 + (1:16),(i - 1) * 16 + (1:16)),image((j - 1) * 16 + (1:1
        r(j,i) = 2/256 + entr(dct_co_qtz((j - 1) * 16 + (1:16),(i - 1) * 16 + (1:16)));
    end
end
```

```
end

function [d,r,rec_image] = copy_mode(image,last_image)

M = 176/16;
N = 144/16;

d = zeros(N,M);
r = zeros(N,M);

% last_image denotes the reconstructed image of the last frame.
rec_image = last_image;

for i = 1:M
    for j = 1:N
        d(j,i) = immse(rec_image((j - 1) * 16 + (1:16),(i - 1) * 16 + (1:16)),image((j - 1) * 16 + (1:1
        r(j,i) = 2/256;
    end
end

end

function [d,r,rec_image] = inter_mode(image,last_image,steplen)

% last_image denotes the reconstructed image of the last frame.
M = 176/16;
N = 144/16;

d = zeros(N,M);
r = zeros(N,M);

[mot_vec, ~, ~] = ME_ES(image, last_image, 16, 10);
frame_pre = motionComp(last_image, mot_vec, 16);
resi = image - frame_pre;

dct_co_res = blkproc(resi, [8 8], @dct2);
dct_co_res_qtzd = midtread(dct_co_res, steplen);
rec_resi = blkproc(dct_co_res_qtzd, [8 8], @idct2);
rec_image = frame_pre + rec_resi;

for i = 1:M
    for j = 1:N
        d(j,i) = immse(rec_image((j - 1) * 16 + (1:16),(i - 1) * 16 + (1:16)), image((j - 1) * 16 + (1:
        r(j,i) = 2/256 + entr(dct_co_res_qtzd((j - 1) * 16 + (1:16),(i - 1) * 16 + (1:16)));
    end
end


end

%% Full Search/Exhaustive Search
function [motionVect,blk_center,costs] = ME_ES(imgP, imgI, mbSize, dm)

    [row, col] = size(imgP);
    blk_center = zeros(2, row*col/(mbSize^2));

    motionVect = zeros(2,row*col/(mbSize^2));
    costs = ones(2*dm+1,2*dm+1)*20000000;
    computations = 0;
```

```
    mb_cnt= 1;
    for i = 1:mbSize:row-mbSize+1
        for j = 1:mbSize:col-mbSize+1
            for m= -dm: dm
                for n= -dm: dm
                    ref_blk_row = i+m;
                    ref_blk_col = j+n;

                    if (ref_blk_row<1||ref_blk_row+mbSize-1>row||ref_blk_col<1||ref_blk_col+mbSize-1>co
                        continue;
                    end

                    costs(m+dm+1,n+dm+1) =...
                        costSAD(imgP(i:i+mbSize-1,j:j+mbSize-1),imgI(ref_blk_row:ref_blk_row+mbSize-1,r
                    computations = computations+1;
                end
            end
            blk_center(1,mb_cnt) = i+ mbSize/2-1;
            blk_center(2,mb_cnt) = j+ mbSize/2-1;
            [dx,dy,~]=minCost(costs);
            motionVect(1,mb_cnt) = dx-dm-1;
            motionVect(2,mb_cnt) = dy-dm-1;
            mb_cnt = mb_cnt+1;
            costs = ones(2*dm+1,2*dm+1)*20000000;
         end
    end
end


function imgComp = motionComp(imgI, motionVect, mbSize)

    [row,col]=size(imgI);
    mb_cnt=1;
    for i = 1:mbSize:row-mbSize+1
        for j = 1:mbSize:col-mbSize+1
             ref_blk_row=i+motionVect(1,mb_cnt);
             ref_blk_col=j+motionVect(2,mb_cnt);
             imgComp(i:i+mbSize-1,j:j+mbSize-1)=imgI(ref_blk_row:ref_blk_row+mbSize-1,ref_blk_col:ref_b
             mb_cnt=mb_cnt+1;
        end
    end
end


function [dx,dy,minc] = minCost(costs)

    minc = min(min(costs));
    [dx, dy] = find(costs == minc);
    dx = dx(1);
    dy = dy(1);
end


function cost = costSAD(currentBlk,refBlk)

    cost=sum(sum((currentBlk-refBlk).^2));
end
```

```
function

function dctblock=dct16(block)
dctblock=blkproc(block,[8 8],@dct2);
end

function idctblock=idct16(block)
idctblock=blkproc(block,[8 8],@idct2);
end

function y = entr(img) % calculate the bit-rate of a given matrix
     [M, N] = size(img);
    [pixel_counts, ~] = groupcounts(img(:)); % find the counting of all levels
    pixel_probas = nonzeros(pixel_counts(:)./(M*N));
    log_pixel_probas = log2(pixel_probas);
    y = -sum(pixel_probas.*log_pixel_probas);

end

function quant = midtread(x,stepsize)
% uniform mid-tread quantizer function
quant = round(x/stepsize)*stepsize;
end

function [r_intra]=RATE(input,stepsize)
[height,width]=size(input);

r_intra=zeros(height/16,width/16)+1/256;
m=1:16;

for row=1:height/16
    for column=1:width/16
qian=dct16(input(16*(row-1)+m,16*(column-1)+m));
hou=midtread(qian,stepsize);
r_intra(row,column)=r_intra(row,column)+entr(hou);
    end
end

end

function [d_copy]=copy_mode_d(input0,input1,stepsize)
[height,width]=size(input1);
d_copy=zeros(height/16,width/16);
%copy mode
m=1:16;
for row=1:height/16
    for column=1:width/16


f_dct=dct16(input0(16*(row-1)+m,16*(column-1)+m));
f_q=midtread(f_dct,stepsize);
f_idct=idct16(f_q);


d_copy(row,column)=immse(f_idct,input1(16*(row-1)+m,16*(column-1)+m));
    end
end
```

```
end

function [d_intra]=intra_mode_d(input,stepsize)
[height,width]=size(input);
d_intra=zeros(height/16,width/16);
m=1:16;
%intra mode
for row=1:height/16
    for column=1:width/16

qian2=dct16(input(16*(row-1)+m,16*(column-1)+m));
hou2=midtread(qian2,stepsize);
d_intra(row,column)=immse(hou2,qian2);
    end
end
end
```

References

[1] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice Hall, 2nd ed., 2002