Project 3: Video Compression

EQ2330 Image and Video Processing, Project 3

Ata Yavuzyilmaz
atay@kth.se

Tom Diener
tdiener@kth.se

December 23, 2022

Summary

The project evaluates three different video compression algorithms regarding their performance and complexity by measuring the rate-PSNR curves for the same video sequences. Concrete, an Intra-Frame Coder, a Conditional Replenishment Coder and a video coder with motion compensation were discussed. For the performed measurements it could be concluded that increasing the complexity of the compression algorithm by adding more modes leads to a better compression result, although demanding more computational resources. It is also observed that different types of videos result in different compression rates, while having the same video quality.

1 Introduction

The aim of this project is to evaluate the achievable performance of three video coding algorithms. The algorithms are Intra-Frame Coder, Conditional Replenishment Coder and Motion Compensated Coder.

Video compression is different from image compression because for videos, there are a lot of frames, which are basically images, that need to be transmitted. The size of high quality videos is too large that it is not possible to transmit them without proper coding algortihms. The algorithms used in this project exploit the similarity between two consecutive frames to reduce the transmitted data.

2 System Description

2.1 Intra-Frame Coder

The Intra-Frame Coder is similar to the 2D-DCT image compression algorithm used in the previous project. The Intra-Frame Coder compresses each frame independently. Hence, it is basically the same system as a DCT image coder. The coder uses 8x8 2D-DCT blocks in this case, however the frame is divided into 16x16 blocks. Therefore, for each block, we apply the 8x8 DCT block for 4 times. After applying the DCT transform, the DCT coefficients are quantized with a uniform scalar quantizer. All of the coefficients are quantized with the same stepsize.

To calculate the distortion, which is the MSE between the original and reconstructed video, Parseval's theorem can be used. The MSE between the original and reconstructed video is equal to the MSE between the original and quantized DCT coefficients.

When transmitting the DCT coefficients, it is assumed that the coder uses variable length coding (VLC) with ideal codeword length. The used VLC is different for the coefficients in the block, but the VLC is the same for the same coefficients in different blocks. Thus, to estimate the bit-rate of this coder, the entropy of these VLCs should be calculated because for the ideal VLCs, the rate is equal to the entropy. The rate R_{intra} is found by calculating the entropy for each coefficient and averaging all of them.

2.2 Conditional Replenishment Video Coder

For videos with large resolution frames, the intra-frame coding will require a lot of rate, which can be impossible to achieve. Therefore, more efficient coding algorithms are needed. The conditional replenishment video coding exploits the similarity of two frames to reduce the rate. This coding algorithm has two modes: *copy mode* and *intra mode.*

For each 16x16 block in the frame, the coder will decide which mode to choose. When the coder is in copy mode, it will copy the coefficients from the same block of the previous frame. In intra mode, the coder applies intra-frame coding.

For each block, the conditional replenishment coder calculates the Lagrangian error function for two modes. The Lagrangian error function is $J_n = D_n + \lambda R_n$, where D_n is the distortion of the frame and R_n is the rate of the frame. The value λ is chosen as $0.0005 \cdot Q^2$. The coder chooses the mode with smaller Lagrangian value. For intra mode, the distortion is the MSE between the original and quantized DCT coefficients. For copy mode, the distortion is the MSE between the previous block's DCT coefficients and current block's coefficients. The transmitter should also send the 1-bit mode information, which is added to both rates. The rates are given as follows:

$$\begin{aligned} R_{intramode} &= 256 \cdot R_{intra} + 1, \\ R_{copymode} &= 1, \end{aligned} \tag{1}$$

where R_{intra} is the rate calculated in Section 2.1. It is multiplied by 256 because R_{intra} is the entropy per pixel. Here we want to calculate the entropy per block.

After choosing the mode for each frame, the distortion and rate values are cumulatively added to obtain a final average distortion and rate for the conditional replenishment video coder. Finally, to find the distortion per pixel, the resulting distortion value is divided to the number of blocks and number of frames. The resulting rate is also divided to the number of frames.

2.3 Video Coder with Motion Compensation

Since usually two consecutive frames of video can contain changes through both moving objects and moving camera position, there could be some redundancy between the images which was not exploited by the copy mode yet. Hence, a third mode named as *intermode* which applies motion compensation can be added to the conditional replenishment video coder and should improve the performance of the compression due to taking more redundancy into account.

In detail, the motion compensation is executed by finding the block of the current frame in the stored reconstructed previous frame. A so called motion estimator function is used here to determine the displacement vector of the block. This is done by iterating over a set of possible displacement vectors within a defined measurement window. All shifted blocks are compared to the original block and the displacement vector for the motion compensated block that minimizes the mean squared error is returned by the motion estimator function.

Nevertheless, the motion compensation itself is not able to perfectly reconstruct the block of the image. For this reason, the residual as the difference between the original and the motion-compensated block is calculated and coded with the DCT-transform as well. This way, the reconstructed image comprises of the the motion-compensated block summed with the residual image.

To estimate the rate of this intermode, equivalently to the first two coding algorithms VLCs with optimal length estimated by the entropy are used. As before, the same coefficients in each block are coded with the same VLC. This requires to determine the statistics of the values for the displacement vectors and the residuals of the whole sequence by performing the inter mode on the whole sequence. Since there are three modes now, two bits are necessary for mode selection which leads to the following rates per block:

$$\begin{aligned} R_{intramode} &= 256 \cdot R_{intra} + 2, \\ R_{copymode} &= 2, \\ R_{intermode} &= R_{motionvector} + 256 \cdot R_{residual} + 2 \end{aligned} \tag{2}$$

Based on this rate, the complete frame sequence can be coded by selecting the mode with the lowest Lagrangian cost for each block. For the Lagrangian, the value λ is chosen as $0.0003 \cdot Q^2$. The distortion is hereby calculated as before as the MSE between the original and the reconstructed block.

3 Results

To understand the success of the coders, the quality of the videos reconstructed at the receiver should be examined depending on the rate. The rate represents how much data is transmitted to send the video through the communication channel. This value shows how much the video is compressed. To calculate the reconstructed video quality, first, the distortion needs to be calculated, which is the MSE between the original video and the reconstructed video. Since MSE takes the average of all the data, the distortion of all the video frames should be averaged. For the reconstructed video quality, the PSNR value can be calculated as

$$PSNR = 10 \log_{10} \left(\frac{255^2}{d} \right) \quad [dB], \tag{3}$$

where d is distortion.

To calculate the bit-rate in terms of kbit/s, the rates per frame are multiplied with the frame rate of 30 frames per second and afterwards the result is divided by 1000 to get the rate in kbit/s.

All the three algorithms are evaluated and compared based on a measured rate-PSNR curve for the video sequences *Foreman* and *Mother & Daughter*. Those curves are measured by calculating the rate and the PSNR of the sequence while varying the stepsize Q of the used quantizer over $Q = 8, 16, 32, 64$. When stepsize changes, the VLCs and rate estimations also change. The results of these measurements are displayed in figure 1 and figure 2. The Conditional Replenishment Coder performs better than the Intra-Frame Coder because it takes advantage of the fact that the background of an image may remain constant for a still image. Thus, adding a copy mode is more efficient in coding the image without a significant loss of quality than coding the whole image each time in intra mode. The figures also show that the video coder with motion compensation improves the performance compared to the Intra-Frame Coder and the Conditional Replenishment Coder as well. The reason for this can be found in exploiting moving a camera position which cannot be compensated only by the copy mode which matches the expectation that an additional mode only selected for improving the performance also improves the performance on the whole video sequence.

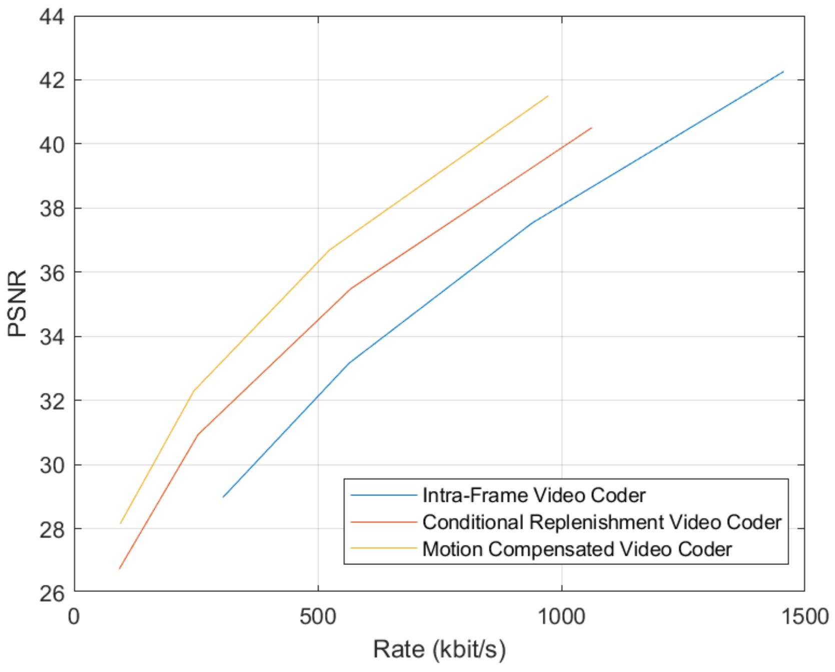

Figure 1: PSNR-bitrate curves for the *Foreman* video.

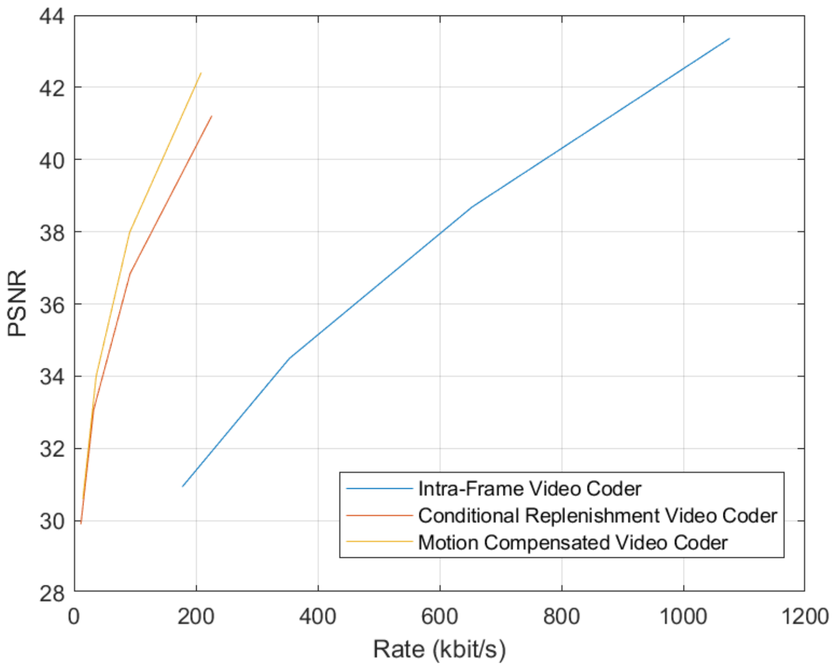

Figure 2: PSNR-bitrate curves for the *Mother & Daughter* video.

When Figure 2 is analyzed, it can be observed that the rates for the Conditional Replenishment and Motion Compensated Video Coder has decreased significantly compared to the Intra-Frame coder. For Figure 1, this difference is lower. The video *Mother & Daughter* is more stable compared to the *Foreman* video. In the *Foreman* video, the background changes while the foreman speaks. However, for the *Mother & Daughter* video, the background is constant, only the Mother and Daughter are moving during the video. Therefore, it can be observed that the Conditional Replenishment and Motion Compensation algorithms worked very efficiently to compress this video, without losing the quality.

4 Conclusions

As it can be seen in the Figures of section 3, with each newly added mode to the coding algorithm, the performance of the coder has improved. This can be explained by the fact that each time some more redundancy is taken into account to increase the compression without suffering a loss of quality.

Furthermore, it is also found that the video characteristics such as the change in the background or change of scenery, highly affects the compression success, regardless of the coding algorithm. The implemented algorithms had more success compressing a video with a constant background.

The parameter in the Lagrangian cost function λ can be used to adjust the trade-off between reduced rate and higher distortion. On the other hand, it is more computationally expensive to make the compression algorithm more complex. A second trade-off between needed computational resources and degree of video compression has to be found.

Appendix

Who Did What

The parts of the project concerning the Intra Frame Video Coder and the Conditional Replenishment Coder were done by Ata Yavuzyilmaz while the video coder with motion compensation was evaluated by Tom Diener.

MatLab code

Main

```
% get first 50 frames of the video
Y = yuv_import_y('foreman_qcif/foreman_qcif.yuv',[176 144],50);
%% Intra-Frame Video Coder
% calculate distortion and rate for different stepsizes
d_intra = zeros(4,1);
en_intra = zeros(4,1);

for i = 3:6
    [d_intra(i-2),en_intra(i-2)] = intraframe_coder(Y, 2^i);
end

% the result is bits/pel, we need kbits/s
[n,m] = size(Y{1});
sz = n*m;
% bit rate in kbit for whole sequence and PSNR for average pixel
entro_intra = en_intra*sz*30/1000;
```

```
psnr_intra = 10*log10(255^2./d_intra);

%% Conditional Replenishment Video Coder
% calculate distortion and rate for different stepsizes
d_cond = zeros(4,1);
en_cond = zeros(4,1);
% measure rate-psnr curve
for i = 3:6
    stepsize = 2^i;
    [d_cond(i-2),en_cond(i-2)] = conditional_rep(Y, stepsize, en_intra);
end
% bit rate in kbit for whole sequence and PSNR for average pixel
entro_cond = en_cond*30/1000;
psnr_cond = 10*log10(255^2./d_cond);

%% Video coder with motion compensation
d_mc = zeros(4,1);
en_mc = zeros(4,1);
% measure rate-psnr curve
for i = 3:6
    stepsize = 2^i;
    [d_mc(i-2),en_mc(i-2)] = coder_mc(Y, stepsize, en_intra);
end
% bit rate in kbit for whole sequence and PSNR for average pixel
entro_mc = en_mc*30/1000;
psnr_mc = 10*log10(255^2./d_mc);

%% plot rate-PSNR curves
figure
plot(entro_intra, psnr_intra)
grid on
hold on
ylabel("PSNR")
xlabel("Rate (kbit/s)")
plot(entro_cond, psnr_cond)
plot(entro_mc, psnr_mc)
legend('Intra-Frame Video Coder','Conditional Replenishment Video Coder','Video coder with
```

Intra-Frame Coder

```
% intra frame video coder function for whole frame sequence
% parameters: video frames and stepsize
% returns: distortion per pixel (mse) and rate (average entropy per pixel)
function [d, entro] = intraframe_coder(Y, stepsize)
    % write functions for 8x8 2d dct
    A = dctmtx(8);
    dct = @(block_struct) A * block_struct.data * A';
    % prepare sorting the coefficients for entropy calculation
    ent = @(block_struct) reshape(transpose(block_struct.data), 1, []);
    % calculate needed parameters of the fram
    num_frames = length(Y);
    [n,m] = size(Y{1});
    frame_size = n*m;

    % 99 blocks from 50 frames (4950), 256 coefficients
    full_VLC = zeros(4950, 256);
    % calculate average distortion
    d = 0;

    % transform frames and get the VLC for all frames
    for i = 1:num_frames
        % transform frame
        transformed = blockproc(Y{i}, [8 8], dct);
        % quantize coefficients
        result = stepsize * round(transformed/stepsize);
        % calculate distortion using Parseval's theorem and add it up
        d = d + immse(result, transformed)/num_frames;
        % prepare entropy
        entro_temp = blockproc(result, [16 16], ent);
        entro_temp = reshape(entro_temp', 256, frame_size/256)';
        % 99 Blocks per image for all frames entro_temp values for all
        % coefficients within one block
        full_VLC(((i-1)*99)+1:99*i,:) = entro_temp;
    end

    % calculate the entropy
    entro = 0;
    % sum and average entropies of all coefficients of all frames to get
    % the entropy per pixel
    for j = 1:256
        % select jth coefficient from all blocks
        sel = full_VLC(:,j);
        entro = entro + entropy(mat2gray(sel))/256;
    end
end
```

Conditional Replenishment Coder

```
% conditional replenishment video coder function
% parameters: video frames, stepsize, entropy of intra frame coding
% returns: distortion (mse) and rate (entropy per frame)
function [d, entro] = conditional_rep(Y, stepsize, en)
num_frames = length(Y);

% write functions for 8x8 2d dct
A = dctmtx(8);
dct = @(block_struct) A * block_struct.data * A';

% for the first frame, apply intra mode (ransform and quantize)
sel = Y{1};
transform = blockproc(sel, [8 8], dct);
quantized = stepsize * round(transform/stepsize);

% calculate distortion for the first frame
d = immse(transform, quantized)/num_frames;
% divide frame to blocks and get parameters for blocks
[rows, cols] = size(sel);
y_l = rows/16;
x_l = cols/16;
```

```
numblocks = x_l * y_l;

% rates for different modes (select to)
R_intra = en(log2(stepsize)-2)*256 + 1;
R_copy = 1;
% calculate the rate for first frame
ent = numblocks * R_intra;

storage = cell(y_l, x_l);
% block by block operation to store the blocks of the first frame
for i = 1:x_l
    for j = 1:y_l
        % flatten the block for easier calculation
        block = quantized(16*(j-1)+1:16*j, 16*(i-1)+1:16*i);
        % store the blocks
        storage{j,i} = block;
    end
end

lambda = 0.0005*(stepsize^2);
% conditional replenishment for other frames
for f = 2:num_frames
    % dct transform of the currnet frame
    sel = Y{f};
    transform = blockproc(sel, [8 8], dct);
    % decide on the mode for each block based on the lagrangian for each
    % mode
    for i = 1:x_l
        for j = 1:y_l
            % make decision on mode

            % get one block of the current frame
            block = transform(16*(j-1)+1:16*j, 16*(i-1)+1:16*i);
            % calculate Lagrangian for copy mode
            d_copy = immse(block, storage{j,i});
            L_copy = d_copy + lambda * R_copy;
            % calculate Lagrangian for intra mode
            quantized = stepsize * round(block/stepsize);
            d_intra = immse(quantized, block);
            L_intra = d_intra + lambda * R_intra;

            % select frame based on minimal value of Lagrangian

            % copy mode
            % update distortion (distortion of first frame through quantization
            % + distortion through copying)
            if L_copy <= L_intra
                d = d + d_copy/(num_frames*numblocks);
                ent = ent + R_copy;
            % intra mode
            else
                % bc loops over all frames and all blocks
                d = d + d_intra/(num_frames*numblocks);
                ent = ent + R_intra;
                % change the stored block
                storage{j,i} = quantized;
            end
        end
    end
end
% summed entropy over all frames has to be averaged for one frame
entro = ent/(num_frames);
end
```

Video Coder with Motion Compensation

```
% Motion compensated video coding
% parameters: video frames and stepsize
% returns: distortion (mse) and rate (entropy per frame)
function [d, entro] = coder_mc(Y, stepsize, en)
num_frames = length(Y);
lambda = 0.0003*(stepsize^2);

% write functions for 8x8 2d dct
A = dctmtx(8);
dct = @(block_struct) A * block_struct.data * A';
% prepare sorting the coefficients for entropy calculation
ent = @(block_struct) reshape(transpose(block_struct.data), 1, []);

% first, apply inter-mode to all frames to get the rate of inter mode
% for the first frame, apply intra mode (transform and quantize)
sel = Y{1};
% divide 1. frame to blocks and get parameters for blocks
[rows, cols] = size(sel);
frame_size = rows*cols;
y_l = rows/16;
x_l = cols/16;
numblocks = x_l * y_l;
% apply intra mode
transform = blockproc(sel, [8 8], dct);
quantized = stepsize * round(transform/stepsize);

% create memory for motion vectors and residual images
motionVec = zeros(2, numblocks, num_frames-1);
% 99 blocks from 49 frames, 256 coefficients
% entropy for residual blocks
residual_VLC = zeros(99*49, 256);

% storage for previous image
storage = quantized;

for f = 2:num_frames
    count = 0;
    % DCT transform of the current frame
    sel = Y{f};
    transform = blockproc(sel, [8 8], dct);
    % residual image
    res_full = zeros(rows,cols);
    % reconstructed image
    recons = zeros(rows,cols);
    % blockwise operation
    for i = 1:x_l
        for j = 1:y_l
            count = count +1;
            % get one block of the current frame
            x_ind = 16*(i-1)+1;
            y_ind = 16*(j-1)+1;
            block = transform(y_ind:y_ind+15, x_ind:x_ind+15);
            % initialize search
            lowest = inf;
            sel_dx = 0;
            sel_dy = 0;
            for dx = -10:10
                for dy = -10:10
                    % check if the block is out of bounds
                    if (x_ind + dx > 0) && (y_ind + dy > 0) && (rows > y_ind + dy ...
                            + 15) && (cols > x_ind + dx + 15)
                        % calculate error
                        err = immse(block, storage(y_ind+dy:y_ind+dy+15,x_ind+dx ...
                            :x_ind+dx+15));
                        % if error is lowest, save the motion vector
                        if err < lowest
                            lowest = err;
                            sel_dx = dx;
                            sel_dy = dy;
                        end
                    end
                end
            end
            % calculate the residual image
            residual = block-storage(y_ind+sel_dy:y_ind+sel_dy+15, ...
                x_ind+sel_dx:x_ind+sel_dx+15);
            % quantize the residual
            residual_q = stepsize * round(residual/stepsize);
            % save the reconstructed block
            res_full(y_ind:y_ind+15, x_ind:x_ind+15) = residual_q;
            recons(y_ind:y_ind+15, x_ind:x_ind+15) = residual_q + ...
            storage(y_ind+sel_dy:y_ind+sel_dy+15,x_ind+sel_dx:x_ind+sel_dx+15);
            % save the motion vectors
            motionVec(:, count, f-1) = [sel_dx, sel_dy];
        end
    end
    % prepare entropy
    entro_temp = blockproc(res_full, [16 16], ent);
    entro_temp = reshape(entro_temp', 256, frame_size/256)';
    residual_VLC(((f-1)*99)+1:99*f,:) = entro_temp;
    % store the current frame
    storage = recons;
end

% calculate the entropy
mot_entro = entropy(mat2gray(motionVec));
res_entro = 0;
```

```
% sum over the entropies of all coefficients of all frames to get the
% total entropy per block
for j = 1:256
    % select jth coefficient from all blocks
    sel = residual_VLC(:,j);
    res_entro = res_entro + entropy(mat2gray(sel))/256;
end

% rates for different modes, since there are 3 modes, 2 bits needed to
% represent the mode
R_intra = en(log2(stepsize)-2)*256 + 2;
% entropy of motion vectors + entropy of residual frame + 2
R_mc = 256*res_entro + mot_entro + 2;
R_copy = 2;

% after calculating the rates, start video coding with motion compensation
% for the first frame, apply intra mode (transform and quantize)
sel = Y{1};
%apply intra mode
transform = blockproc(sel, [8 8], dct);
quantized = stepsize * round(transform/stepsize);

% calculate distortion for the first frame
d = immse(transform, quantized)/num_frames;
% calculate the rate for first frame
ent = numblocks * R_intra;

% storage for previous image
storage = quantized;

% code all frames
for f = 2:num_frames
    % dct transform of the current frame
    sel = Y{f};
    transform = blockproc(sel, [8 8], dct);
    % reconstructed image
    recons = zeros(rows,cols);
    % decide on the mode for each block based on the lagrangian for each
    % mode
    for i = 1:x_l
        for j = 1:y_l
            % get one block of the current frame
            x_ind = 16*(i-1)+1;
            y_ind = 16*(j-1)+1;
            block = transform(y_ind:y_ind+15, x_ind:x_ind+15);
            % initialize search
            lowest = inf;
            sel_dx = 0;
            sel_dy = 0;
            for dx = -10:10
                for dy = -10:10
                    % check if the block is out of bounds
                    if (x_ind + dx > 0) && (y_ind + dy > 0) && (rows > y_ind + dy ...
                            + 15) && (cols > x_ind + dx + 15)
                        % calculate error
                        err = immse(block, storage(y_ind+dy:y_ind+dy+15,x_ind+dx ...
                            :x_ind+dx+15));
                        % if error is lowest, save the motion vector
                        if err < lowest
                            lowest = err;
                            sel_dx = dx;
                            sel_dy = dy;
                        end
                    end
                end
            end
            % calculate the residual image
            residual = block-storage(y_ind+sel_dy:y_ind+sel_dy+15, ...
                x_ind+sel_dx:x_ind+sel_dx+15);
            % quantize the residual
            residual_q = stepsize * round(residual/stepsize);
            % calculate the reconstructed block
            recons_mc = residual_q + ...
            storage(y_ind+sel_dy:y_ind+sel_dy+15,x_ind+sel_dx:x_ind+sel_dx+15);

            % make decision on mode
            % calculate Lagrangian for copy mode
            d_copy = immse(block, storage(y_ind:y_ind+15,x_ind:x_ind+15));
            L_copy = d_copy + lambda * R_copy;

            % calculate Lagrangian for mc mode
            d_mc = immse(block, recons_mc);
            L_mc = d_mc + lambda * R_mc;

            % calculate Lagrangian for intra mode
            quantized = stepsize * round(block/stepsize);
            d_intra = immse(quantized, block);
            L_intra = d_intra + lambda * R_intra;

            % select frame based on minimal value of Lagrangian
            L = [L_copy, L_mc, L_intra];
            [~, ind] = min(L);

            if ind == 2
                % mc mode
                d = d + d_mc/(num_frames*numblocks);
                ent = ent + R_mc;
                recons(y_ind:y_ind+15, x_ind:x_ind+15) = recons_mc;
            elseif ind == 3
                % intra mode
                % bc loops over all frames and all blocks
                d = d + d_intra/(num_frames*numblocks);
                ent = ent + R_intra;
                % change the stored block
                recons(y_ind:y_ind+15, x_ind:x_ind+15) = quantized;
            else
                % copy mode
                d = d + d_copy/(num_frames*numblocks);
```

```
                ent = ent + R_copy;
                recons(y_ind:y_ind+15, x_ind:x_ind+15) = ...
                storage(y_ind:y_ind+15,x_ind:x_ind+15);
            end
        end
    end
    storage = recons;
end
% summed entropy over all frames has to be averaged for one frame
entro = ent/(num_frames);
end
```

References

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