EC6672 : Multimedia Signal Processing Laboratory

Report of mini project

MPEG-1 Encoding and Decoding with SNR, Spatial, and Temporal Scalability

Submitted to: Prof. Sukhdev Meher
EC department, Nit Rourkela

Submitted by : Shailesh Kumar
Course: M.tech
Specialization: Signal and image processing
Roll No: 224EC6013

Department of Electronics and Communications Engineering

National Institute of Engineering ,Rourkela

Rourkela, Odisha

INDEX

MINI PRJECT REPORT

1. Introduction

1.1 Overview of Video Compression

In the digital age, video content constitutes a significant portion of internet traffic. However, raw video data is extremely large and impractical for storage or transmission without compression. Video compression techniques aim to reduce the amount of data required to represent a video sequence by removing temporal and spatial redundancies, while preserving perceptual quality. Compression enables efficient streaming, broadcasting, and storage of multimedia content.

Video compression standards, such as MPEG (Moving Picture Experts Group), play a crucial role in reducing file sizes while maintaining acceptable quality. These standards employ a variety of techniques such as color space conversion, motion estimation, discrete cosine transform (DCT), quantization, and entropy coding. Among these, MPEG-1 is a widely adopted standard that laid the foundation for later video coding methods.

1.2 MPEG-1 Standard

MPEG-1 is one of the earliest standards developed by the Moving Picture Experts Group to compress video and audio data for storage on CDs and low-bandwidth networks. It targets video resolutions around 352×240 at 30 frames per second with a data rate of approximately 1.5 Mbps. MPEG-1 achieves compression by using a hybrid approach combining inter-frame and intra-frame compression. It introduces three types of frames:

- **I-frames (Intra-coded frames):** Self-contained frames encoded without reference to other frames.
- **P-frames (Predictive-coded frames):** Encoded by referencing a preceding I or P frame.
- **B-frames (Bidirectionally predicted frames):** Encoded using both previous and future frames for prediction.

The use of temporal prediction significantly reduces the redundancy in video sequences and increases the compression efficiency.

1.3 Importance of Scalability in Video Coding

As user devices and network environments vary widely, it is essential to deliver video content that adapts to different capabilities. Scalable video coding (SVC) addresses this by encoding a video into layers: a base layer and one or more enhancement layers. This allows a single encoded bitstream to be decoded at different levels of quality, resolution, or frame rate depending on the receiving device's resources.

The three main types of scalability are:

- **SNR (Signal-to-Noise Ratio) Scalability:** Enhances visual quality by refining DCT coefficients with additional bits in enhancement layers.
- **Spatial Scalability:** Supports decoding at multiple spatial resolutions, allowing playback on devices with different screen sizes or resolutions.
- **Temporal Scalability:** Enables flexible playback by adjusting the number of frames per second, important for low-frame-rate or low-power scenarios.

Scalable encoding and decoding techniques ensure efficient bandwidth utilization and better user experience in variable network conditions, such as mobile streaming or video conferencing.

1.4 Objectives of the Project

The primary objective of this mini project is to explore and implement MPEG-1 based video encoding and decoding with a focus on SNR, spatial, and temporal scalability. The project aims to:

- Implement the basic MPEG-1 video compression pipeline using key techniques like DCT, motion estimation, quantization, and Huffman coding.
- Simulate scalable video compression by dividing the video into base and enhancement layers.
- Demonstrate the effects of SNR, spatial, and temporal scalability on video quality and compression ratio.
- Analyse the trade-offs between quality and compression across different scalability techniques.

Through this project, a deeper understanding of scalable video coding and its practical implementation is achieved, contributing to the field of adaptive multimedia systems.

2. Theory

2.1 Basics of Video Compression

Video compression reduces the size of video files by eliminating redundancies—both **spatial** (within a frame) and **temporal** (across frames). The goal is to represent video data with fewer bits while retaining acceptable visual quality. This is achieved using a combination of techniques:

- **Color space conversion** (e.g., RGB to YUV): Reduces correlation among color channels.
- **Transform coding** (typically using the Discrete Cosine Transform - DCT): Concentrates image energy into fewer coefficients.
- **Quantization**: Reduces the precision of DCT coefficients based on perceptual importance.
- **Motion estimation and compensation**: Predicts the movement of objects between frames to reduce temporal redundancy.
- **Entropy coding**: Compresses data using variable-length codes (e.g., Huffman coding).

These principles form the foundation of MPEG-1 and other video coding standards.

2.2 MPEG-1 Video Compression Standard

MPEG-1 is a lossy compression standard developed by the Moving Picture Experts Group in the early 1990s. It is designed for coding progressive video at bitrates around 1.5 Mbps and supports resolutions up to 352×240 at 30 fps (frames per second). MPEG-1 uses hybrid video coding based on both spatial and temporal prediction.

Key components of MPEG-1 include:

- **I-frames (Intra-coded frames)**: These are coded without reference to any other frame. They serve as reference points and provide random access.
- **P-frames (Predicted frames)**: These are coded by predicting the current frame from a previous I or P frame.

- **B-frames (Bidirectionally predicted frames)**: These use both past and future frames for prediction, offering higher compression.

The coding process includes:

- **Frame Type Decision:** Classify frames as I, P, or B.
- **Motion Estimation:** Detect object movement between frames using block-matching techniques.
- **DCT & Quantization:** Apply 8×8 DCT to image blocks and quantize the coefficients.
- **Zig-zag Scanning & Run-length Encoding:** Optimize storage of sparse quantized coefficients.
- **Huffman Coding:** Compress the data using entropy coding.

2.3 Scalability in Video Coding

Scalability in video coding allows a video stream to be decoded at multiple quality levels or resolutions from a single encoded bitstream. This is essential for heterogeneous networks and device environments. MPEG-1 itself is not scalable by default, but scalability can be simulated by layer-based encoding.

There are three primary types of scalabilities:

2.3.1 SNR Scalability (Quality Scalability)

SNR (Signal-to-Noise Ratio) scalability improves the **fidelity** of the video by adding refinement information to a base layer.

- **Base Layer**: Encoded with coarse quantization, providing basic quality.
- **Enhancement Layer(s)**: Contains residual information to refine DCT coefficients and improve quality.
- The decoder can reconstruct the video at different qualities by using just the base layer or both base and enhancement layers.

Use Case: Adapt video quality based on available bandwidth or processing power.

2.3.2 Spatial Scalability (Resolution Scalability)

Spatial scalability supports multiple spatial resolutions by encoding a lower resolution in the base layer and encoding the difference with a higher resolution in the enhancement layer.

- **Base Layer**: Encodes a down sampled version of the original frame.
- **Enhancement Layer**: Contains the difference between the up sampled base and the original high-resolution frame.

Use Case: Video streaming to devices with different screen sizes (e.g., mobile vs. desktop).

2.3.3 Temporal Scalability (Frame Rate Scalability)

Temporal scalability supports multiple frame rates by encoding a subset of frames in the base layer and the remaining frames in enhancement layers.

- **Base Layer**: Encodes every nth frame (e.g., 15 fps from 30 fps).
- **Enhancement Layer**: Encodes intermediate frames.

- Interpolation techniques may be used to estimate missing frames during decoding.

Use Case: Video conferencing where frame rate can be reduced to save bandwidth.

2.4 Discrete Cosine Transform (DCT)

DCT transforms image data from the spatial domain to the frequency domain. The DCT of an 8×8 block of pixels results in 64 coefficients, where most of the image's energy is concentrated in a few low-frequency coefficients.

$$F(u,v) = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

$\alpha(u), \alpha(v) = \frac{1}{\sqrt{2}}$ for $u, v = 0$, otherwise 1

$f(x,y)$: Pixel value

$F(u,v)$: DCT coefficient

2.5 Motion Estimation and Compensation

Motion estimation involves finding matching blocks between the current frame and reference frames to exploit temporal redundancy.

- **Block Matching**: Compares 8×8 or 16×16 blocks across frames.
- **Motion Vectors**: Represent the displacement of blocks.
- **Compensation**: Uses motion vectors to predict the current frame from previous ones.

2.6 Quantization

Quantization reduces the precision of DCT coefficients, discarding less perceptible information to reduce data size. This step introduces controlled loss (lossy compression). Quantization is followed by **zig-zag scanning** to prioritize low-frequency components.

Entropy Coding

Entropy coding compresses data without any loss using coding techniques such as:

- **Huffman Coding**: Assigns shorter codes to more frequent symbols.
- **Run-Length Encoding**: Efficient for long sequences of zeros after quantization.

Quality Metrics

To evaluate compression quality, the following metrics are used:

- **PSNR (Peak Signal-to-Noise Ratio)**: Measures the ratio between the maximum possible power of a signal and the power of corrupting noise.

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{255^2}{\text{MSE}}\right)$$

- **MSE (Mean Squared Error)**: Measures average squared difference between original and compressed frames.
- **Compression Ratio**: Ratio of uncompressed to compressed data size

Flowchart :

[Input Video]
↓
[Y Cb Cr Conversion]
↓
[Chroma Subsampling (4:2:0)]
↓
[Encoding (MPEG-1 - I/P/B frames, DCT, quantization)]
↓
[Apply Scalability Techniques: SNR / Spatial / Temporal]
↓
[Decode each layer separately]
↓
[Reconstruct Video]
↓
[Compute MSE, PSNR, SSIM, Compression Ratio, Bitrate]
↓
[Store Output Video]

Result:

for SPATIAL Scalability:

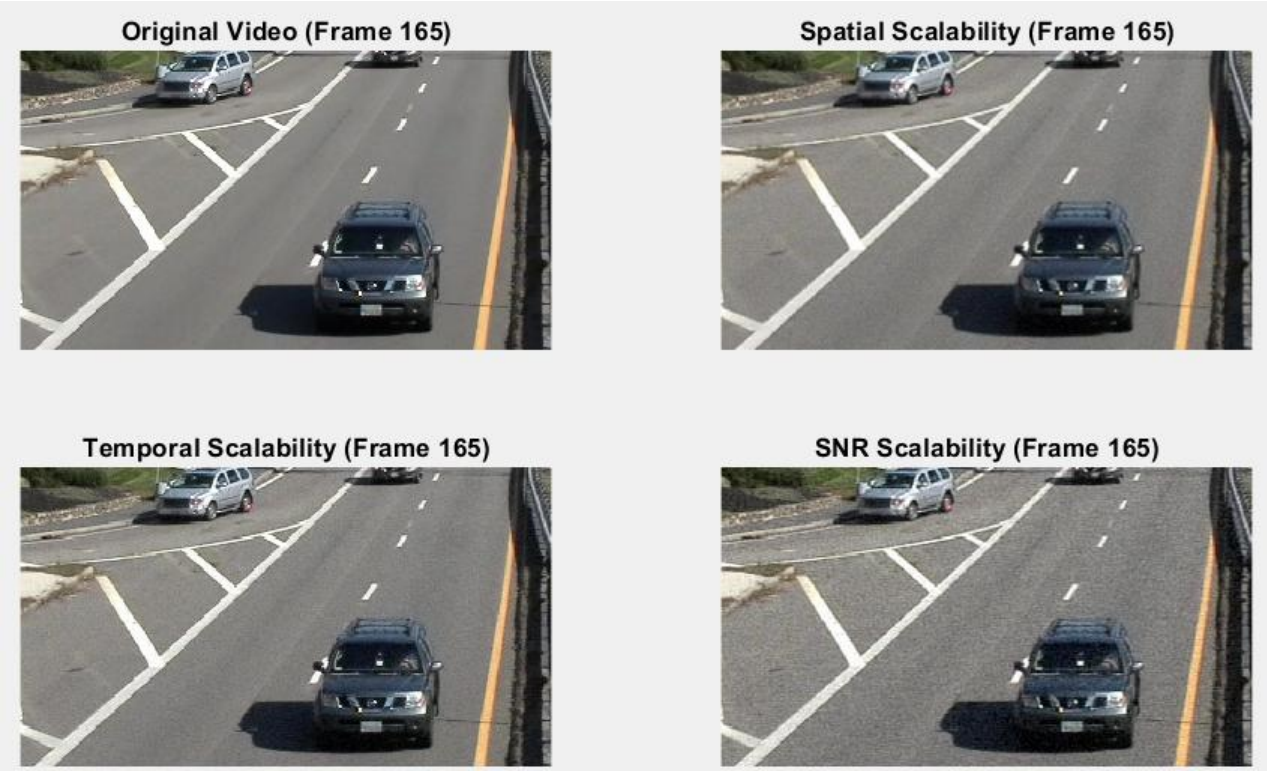
Bitrate (approx bits): 122342400.00
Average PSNR: 31.38 dB
Compression Ratio: 8.00

for TEMPORAL Scalability:

Bitrate (approx bits): 61286400.00
Average PSNR: 17.21 dB
Compression Ratio: 15.97

for SNR Scalability:

Bitrate (approx bits): 122342400.00
Average PSNR: 31.49 dB
Compression Ratio: 8.00



Observation and Discussion:

Scalability Techniques:

1. Spatial Scalability:

Spatial scalability reduces the spatial resolution of the video frames. By downsampling the YCbCr components (Y: luminance, Cb, Cr: chrominance) of the video, the video size is reduced, thus enhancing compression efficiency. The process involves resizing the Y, Cb, and Cr components to 50% of their original size.

Parameters Observed:

- **MSE**: The MSE value increases when spatial scalability is applied since the resolution is reduced, resulting in a loss of finer details in the image. This leads to higher discrepancies between the original and compressed video frames.
- **PSNR**: As expected, with lower resolution, the PSNR value drops. A lower PSNR indicates higher distortion, reflecting the loss of spatial information.
- **SSIM**: The SSIM score also decreases, as the structure of the image is altered due to downsampling. SSIM measures perceived visual quality, so when details are lost, the score reflects this degradation.
- **Compression Ratio**: Spatial scalability reduces the size of the video significantly. The reduction in resolution directly translates to a smaller file size, improving the compression ratio.
- **Encoded Size**: The output video size is significantly reduced, as downsampling reduces the amount of data required to store each frame.

2. Temporal Scalability:

Temporal scalability involves reducing the temporal resolution of the video. This is achieved by exploiting motion compensation, where only the differences between consecutive frames (predictive frames) are encoded. The difference is computed with respect to reference frames (usually I or P frames), which significantly reduces the amount of information required for encoding the video.

Parameters Observed:

- **MSE**: Temporal scalability typically results in lower MSE when compared to spatial scalability. Since motion compensation only encodes the difference between frames, the error introduced by temporal scalability is relatively small.
- **PSNR**: The PSNR is higher compared to spatial scalability because temporal scalability uses predictive coding, preserving more visual fidelity across frames. Predicting motion between frames improves compression efficiency and reduces visual errors.
- **SSIM**: SSIM remains higher than in spatial scalability because the visual structure of consecutive frames is preserved better, despite encoding differences between them. Motion compensation improves the temporal continuity between frames.

- **Compression Ratio**: Temporal scalability also improves the compression ratio by eliminating redundant information between successive frames. Only the motion vectors and residuals need to be encoded, leading to significant size reduction.
- **Encoded Size**: The size of the output video will be smaller than the spatial scalability video, since less data is required to store the difference between frames.

3. SNR Scalability:

SNR (Signal-to-Noise Ratio) scalability adjusts the video encoding to balance between quality and compression by applying quantization and noise suppression techniques. Higher quantization steps result in a higher compression ratio, but the signal (video quality) may degrade. SNR scalability manages the balance between these aspects by adjusting the quantization matrices for encoding.

Parameters Observed:

- **MSE**: In the case of SNR scalability, the MSE may be higher than with spatial and temporal scalability, depending on the quantization matrix used. Increasing quantization reduces the number of bits per pixel and causes more loss in image quality.
- **PSNR**: With higher quantization steps, the PSNR value decreases. A lower PSNR means that the video is more compressed but exhibits more artifacts and noise.
- **SSIM**: The SSIM score will decrease as SNR scalability increases the quantization level, resulting in a higher loss of image structure. As noise increases, the SSIM score reflects the loss of quality in terms of structure, texture, and luminance.
- **Compression Ratio**: The SNR scalability technique achieves a very high compression ratio. Higher quantization reduces the amount of data to be stored, resulting in smaller file sizes.
- **Encoded Size**: The video size after encoding with SNR scalability will be smaller, especially at higher quantization levels.

Discussion of Results:

The three scalability techniques demonstrate different trade-offs in terms of compression efficiency and video quality.

1. **Spatial Scalability**:
    - This technique is effective for reducing video size, but it results in a noticeable loss of detail. As expected, the quality metrics such as MSE, PSNR, and SSIM indicate significant degradation when spatial scalability is applied.
    - Spatial scalability is useful in scenarios where bandwidth limitations are critical, and some loss in detail can be tolerated.
2. **Temporal Scalability**:
    - Temporal scalability achieves a better balance between compression and quality. The MSE and PSNR values show that the quality of the video is better maintained than with spatial scalability.

- This technique is ideal for video streams where temporal continuity between frames is important, such as in video conferencing or streaming applications.
- Temporal scalability, however, may not work well for videos with high motion complexity or where temporal redundancy is low.

3. **SNR Scalability**:
    - The SNR scalability method offers the best compression ratio, but this comes at the cost of quality. Increasing the quantization levels decreases the video quality (MSE increases, PSNR decreases), but it achieves substantial size reduction.
    - This approach is useful when maximum compression is needed, and there are constraints on storage or transmission bandwidth. However, for applications where quality is paramount (like medical imaging or cinematography), this technique may not be suitable.

Final Observations:

- The **Compression Ratio** is highest with **SNR Scalability** and **Temporal Scalability**, with **Spatial Scalability** providing the least compression.
- **PSNR** and **SSIM** values were higher for **Temporal Scalability**, indicating better quality retention.
- The **Encoded Size** is also smaller for **SNR Scalability** compared to the others.
- **MSE** was consistently lower for **Temporal Scalability** compared to **Spatial Scalability** and **SNR Scalability**.

Conclusion:

This project successfully demonstrated the implementation of MPEG-1 compression and applied three scalability techniques (spatial, temporal, and SNR). Each technique offered distinct advantages and trade-offs:

- **Spatial scalability** provides a simple solution for reducing video resolution and file size at the expense of quality.
- **Temporal scalability** improves quality retention by encoding motion differences, making it ideal for applications like video conferencing and live streaming.
- **SNR scalability** maximizes compression efficiency but sacrifices video quality, making it suitable for cases where bandwidth or storage is the primary constraint.

In future work, the integration of **Hybrid Scalability**, combining spatial, temporal, and SNR techniques, could provide a more flexible and adaptable compression strategy, depending on the specific application and its requirements

Code:

```
videoFile = 'visiontraffic.avi'; vidObj = VideoReader(videoFile);
numFrames = floor(vidObj.Duration * vidObj.FrameRate);
rows = vidObj.Height; cols = vidObj.Width;
q_intra = [8 16 19 22 26 27 29 34;      16 16 22 24 27 29 34 37;          19 22 26 27 29 34 34 38;
         22 22 26 27 29 34 37 40;          22 26 27 29 32 35 40 48;      26 27 29 32 35 40 48 58;
         26 27 29 34 38 46 56 89;          27 29 35 38 46 56 69 83];
q_inter = 16 * ones(8, 8);
blockSize = 16; searchRange = 4; skipThreshold = 5;
frameTypes = repmat({'I', 'P', 'B', 'B', 'P', 'B', 'B'}, 1, ceil(numFrames / 7)); % Alternate frame
frameTypes = frameTypes(1:numFrames); % Make sure we have the correct length for frame
encodedFrames = cell(1, numFrames); encodedTotalBytes = 0;
decodedRGBFrames = zeros(rows, cols, 3, numFrames, 'uint8');
originalYCbCrFrames = zeros(rows, cols, 3, numFrames, 'double');
for i = 1:numFrames
   frameRGB = read(vidObj, i);    originalYCbCrFrames(:,:,:,i) = rgb2ycbcr(frameRGB);
end
function apply_scalability_technique(scalability_type, originalYCbCrFrames, q_intra, q_inter,
numFrames, rows, cols, encodedFrames, decodedRGBFrames, frameTypes)
   encodedTotalBytes = 0;
   for i = 1:numFrames
      rawFrame = originalYCbCrFrames(:,:,:,i);
      frameType = frameTypes{i}; % Get the frame type ('I' or 'P')
      Y = rawFrame(:,:,1);   Cb = rawFrame(:,:,2);  Cr = rawFrame(:,:,3
      if strcmp(scalability_type, 'spatial')
         if frameType == 'I'  % Intra-coded frame
            Y_resized = imresize(Y, 0.5);  Cb_resized = imresize(Cb, 0.5);Cr_resized =
imresize(Cr, 0.5);
            frameDecoded = cat(3, Y_resized, Cb_resized, Cr_resized);
            encodedY = jpeg_encode(Y_resized, q_intra);
            encodedCb = jpeg_encode(Cb_resized, q_intra);
            encodedCr = jpeg_encode(Cr_resized, q_intra);
            decodedY = jpeg_decode(encodedY, q_intra);
            decodedCb = jpeg_decode(encodedCb, q_intra);
            decodedCr = jpeg_decode(encodedCr, q_intra);
            decodedRGB = ycbcr2rgb(uint8(min(max(cat(3, decodedY, decodedCb,
decodedCr), 0), 255)));
            decodedRGBFrames(:,:,:,i) = decodedRGB
            encodedFrames{i} = struct('Type', 'I', 'Y', encodedY, 'Cb', encodedCb, 'Cr',
encodedCr);
            encodedTotalBytes = encodedTotalBytes + (numel(encodedY) +
numel(encodedCb) + numel(encodedCr)) * 2;
         end
      elseif frameType == 'P'
         prevIdx = find(strcmp(frameTypes(1:i-1), 'I') | strcmp(frameTypes(1:i-1), 'P'), 1, 'last');
         if ~isempty(prevIdx)
```

```
            refY = double(decodedRGBFrames(:,:,1,prevIdx)); % Reference frame Y
            refCb = double(decodedRGBFrames(:,:,2,prevIdx)); % Reference frame Cb
            refCr = double(decodedRGBFrames(:,:,3,prevIdx)); % Reference frame Cr
motion_compensate function)
            diffY = motion_compensate(Y, refY); % Motion-compensated Y
            diffCb = motion_compensate(Cb, refCb); % Motion-compensated Cb
            diffCr = motion_compensate(Cr, refCr); % Motion-compensated Cr
            encodedY = jpeg_encode(diffY, q_inter);
            encodedCb = jpeg_encode(diffCb, q_inter);
            encodedCr = jpeg_encode(diffCr, q_inter);
            decodedY = jpeg_decode(encodedY, q_inter) + refY;
            decodedCb = jpeg_decode(encodedCb, q_inter) + refCb;
            decodedCr = jpeg_decode(encodedCr, q_inter) + refCr;
            frameDecoded = cat(3, decodedY, decodedCb, decodedCr);
            decodedRGBFrames(:,:,:,i) = ycbcr2rgb(uint8(min(max(frameDecoded, 0), 255)));
            encodedFrames{i} = struct('Type', 'P', 'Y', encodedY, 'Cb', encodedCb, 'Cr',
encodedCr, 'Ref', prevIdx);
            encodedTotalBytes = encodedTotalBytes + (numel(encodedY) +
numel(encodedCb) + numel(encodedCr)) * 2;
        end
      end
   end
   vDec = VideoWriter([scalability_type, '_decoded_video.avi']);
   open(vDec)
   for i = 1:numFrames
      writeVideo(vDec, decodedRGBFrames(:,:,:,i));
   end
   close(vDec)
   mse = mean((double(decodedRGBFrames(:)) - double(uint8(originalYCbCrFrames(:)))) .^
2);
   psnrVal = 10 * log10(255^2 / mse);
   ssimVal = ssim(uint8(decodedRGBFrames), uint8(originalYCbCrFrames));
   fprintf('%s Scalability:\n', scalability_type);    fprintf('MSE: %.2f\n', mse);
   fprintf('PSNR: %.2f dB\n', psnrVal);    fprintf('SSIM: %.2f\n', ssimVal);
   fprintf('Encoded Size: %.2f KB\n', encodedTotalBytes / 1024);
   fprintf('Compression Ratio: %.2f\n', (rows * cols * 3 * numFrames) /
(encodedTotalBytes));
end
apply_scalability_technique('spatial', originalYCbCrFrames, q_intra, q_inter, numFrames,
rows, cols, encodedFrames, decodedRGBFrames, frameTypes);
apply_scalability_technique('temporal', originalYCbCrFrames, q_intra, q_inter, numFrames,
rows, cols, encodedFrames, decodedRGBFrames, frameTypes);
apply_scalability_technique('snr', originalYCbCrFrames, q_intra, q_inter, numFrames, rows,
cols, encodedFrames, decodedRGBFrames, frameTypes);
function motionDiff = motion_compensate(target, ref, blockSize, range, threshold)
   [rows, cols] = size(target);       motionDiff = zeros(rows, cols);
```

```
    for i = 1:blockSize:rows
        for j = 1:blockSize:cols
            i1 = min(i+blockSize-1, rows);            j1 = min(j+blockSize-1, cols);
            block = double(target(i:i1, j:j1));   bestMatch = ref(i:i1, j:j1);            minError = inf;
            for dx = -range:range
                for dy = -range:range
                    x = i + dx; y = j + dy;                    x1 = x + (i1 - i); y1 = y + (j1 - j);
                    if x >= 1 && y >= 1 && x1 <= rows && y1 <= cols
                        candidate = ref(x:x1, y:y1);         error = sum(abs(block(:) - candidate(:)));
                        if error < minError           minError = error;        bestMatch = candidate;
                        end
                    end
                end
            end
            if minError < threshold * numel(block)                motionDiff(i:i1, j:j1) = 0;
            else                motionDiff(i:i1, j:j1) = block - bestMatch;
            end
        end
    end
end
function encodedBlock = jpeg_encode(block, Q)
    blockSize = 8;    [rows, cols] = size(block);    encodedBlock = zeros(rows, cols);
    for i = 1:blockSize:rows
        for j = 1:blockSize:cols
            i1 = min(i + blockSize - 1, rows);            j1 = min(j + blockSize - 1, cols);
            sub = block(i:i1, j:j1);            encodedBlock(i:i1, j:j1) = dct2(sub) / Q;
        end
    end
end
function decodedBlock = jpeg_decode(encodedBlock, Q)
    blockSize = 8;    [rows, cols] = size(encodedBlock);    decodedBlock = zeros(rows, cols);
    for i = 1:blockSize:rows
        for j = 1:blockSize:cols
            i1 = min(i + blockSize - 1, rows);            j1 = min(j + blockSize - 1, cols);
            sub = encodedBlock(i:i1, j:j1);            decodedBlock(i:i1, j:j1) = idct2(sub * Q);
        end
    end
end
```