Experiment No: 8

Aim: Study and design of video compression with MPEG-1 standard.

Software Used: MATLAB R2024b.

Theory:

MPEG (Moving Picture Experts Group) is a standard for compressing audio and video. MPEG-1 is one of the early standards designed for compressing VHS-quality video and CD audio to a bitrate of about 1.5 Mbps without a significant loss in quality. It is widely used in Video CDs and some online video formats.

MPEG achieves compression through a combination of techniques such as spatial compression (intra-frame) and temporal compression (inter-frame), along with motion estimation and quantization. It significantly reduces the data rate while maintaining acceptable visual quality.
MPEG uses three types of frames to optimize compression:

I-Frame (Intra-coded Frame)

- Encoded using only information within the frame.
- Serves as a reference for other frames (keyframes).
- Compressed using spatial compression methods like JPEG (DCT and quantization).
- No motion compensation.

P-Frame (Predictive-coded Frame)

- Encoded using motion-compensated prediction from the previous I- or P-frame.
- Stores only the difference (residual) between the current frame and predicted frame.
- Uses motion vectors and residual encoding.
- More compressed than I-frames.

B-Frame (Bi-directionally Predictive-coded Frame)

- Uses both previous and next I- or P-frames for prediction.
- Most compressed frame type.
- Requires decoding of both past and future frames.
- Not used as a reference frame.

Encoding Process:

- **Convert RGB to YCbCr:** Separates brightness (Y) and color (Cb, Cr) components.
- **Chroma Subsampling (4:2:0):**Reduces color data (Cb and Cr) to compress more efficiently.
- **Frame Type Assignment:**Frames are marked as I, P, or B based on prediction strategy.
- **Motion Estimation (for P and B frames):**ompares with reference frames to compute motion vectors and differences.
- **Block Division (8x8):**Each frame or difference frame is divided into 8×8 blocks.
- **Apply DCT (Discrete Cosine Transform):**Converts spatial image blocks to frequency domain.
- **Quantization using Q-table:**Reduces precision of DCT coefficients to compress data.
- **Entropy Coding (Huffman/Run-length(optional)):**Compresses quantized data based on repetition and frequency.

Decoding Process:

- **Entropy Decoding:** Decodes Huffman codes to get quantized DCT coefficients.
- **Inverse Quantization:**Multiplies quantized coefficients with the Q-table.
- **Inverse DCT:** Converts frequency data back to spatial image blocks.

- **Motion Compensation (for P and B frames):** Uses motion vectors and reference frames to reconstruct the image.
- **Reconstruct Frame:** Adds residual to reference prediction for P and B frames.
- **Chroma Upsampling (4:2:0 → 4:4:4):** Restores full-size Cb and Cr components.
- **Convert YCbCr to RGB:** Final reconstructed frame is converted back for display.

Result:

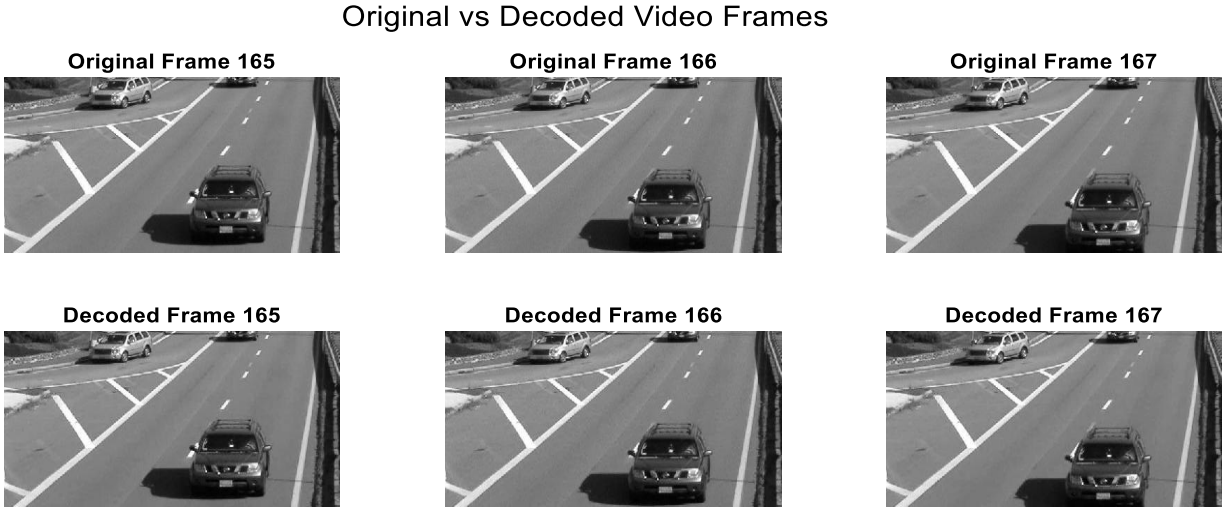
Raw Size: 119475.00 KB

Encoded Size: 26479.56 KB

Compression Ratio: 4.51

MSE: 216.98

PSNR: 24.77 dB



Original video **Decoded video**

Discussion:

- Intra-coded (I) frames are self-contained and use JPEG-like compression; thus, they are larger but provide reference quality.
- Predicted (P) frames use motion estimation from the last I or P frame, significantly reducing size by only encoding differences.
- Bidirectionally predicted (B) frames use both past and future reference frames, allowing maximum data reuse and minimal size.
- Accurately estimated motion vectors in P and B frames result in better prediction and higher compression.
- I > P > B in terms of size, with B-frames being smallest due to dual-frame referencing.
- Reduces color resolution with minimal visible impact, further lowering data size.
- Higher quantization results in more compression but visible artifacts, especially in high-frequency regions.
- Using motion compensation eliminates repetitive encoding of similar objects across frames.

- Since B-frames require both previous and future reference frames, they introduce processing delay in real-time scenarios.
- MPEG relies on a consistent I-P-B pattern (e.g., I-B-B-P...) for optimal balance between quality and compression.

Conclusion:

In this experiment, MPEG video compression was implemented using a combination of I, P, and B frames. I-frames provided high-quality reference frames using JPEG encoding, while P and B frames utilized motion estimation for inter-frame compression, significantly reducing data redundancy. The use of chroma subsampling (4:2:0) and quantization effectively reduced file size with acceptable visual quality. It was observed that B-frames offered the highest compression, though at the cost of increased complexity and delay. Overall, the MPEG approach demonstrates an efficient balance between video quality and compression, making it suitable for practical multimedia applications.

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','B','B','P','B','B','P','B','B'}, 1, ceil(numFrames/9));
frameTypes = frameTypes(1:numFrames);
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
for i = 1:numFrames
   rawFrame = originalYCbCrFrames(:,:,:,i);    frameType = frameTypes{i};
   Y = rawFrame(:,:,1);    Cb = rawFrame(:,:,2);    Cr = rawFrame(:,:,3);
   if frameType == 'I'
      encodedY = jpeg_encode(Y, q_intra);       encodedCb = jpeg_encode(Cb, q_intra);
      encodedCr = jpeg_encode(Cr, q_intra);       decodedY = jpeg_decode(encodedY, q_intra);
      decodedCb = jpeg_decode(encodedCb, q_intra);   decodedCr = jpeg_decode(encodedCr, q_intra);
      encodedFrames{i} = struct('Type', 'I', 'Y', encodedY, 'Cb', encodedCb, 'Cr', encodedCr);
      frameDecoded = cat(3, decodedY, decodedCb, decodedCr);
      decodedRGBFrames(:,:,:,i) = ycbcr2rgb(uint8(min(max(frameDecoded, 0), 255)));
      encodedTotalBytes = encodedTotalBytes + (nnz(encodedY) + nnz(encodedCb) + nnz(encodedCr))
* 2;
   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));
            refCb = double(decodedRGBFrames(:,:,2,prevIdx));
            refCr = double(decodedRGBFrames(:,:,3,prevIdx));
            diffY = motion_compensate(Y, refY, blockSize, searchRange, skipThreshold);
            diffCb = motion_compensate(Cb, refCb, blockSize, searchRange, skipThreshold);
            diffCr = motion_compensate(Cr, refCr, blockSize, searchRange, skipThreshold);
            encodedY = jpeg_encode(diffY, q_inter); decodedY = jpeg_decode(encodedY, q_inter) + refY;
            encodedCb = jpeg_encode(diffCb, q_inter);
            decodedCb = jpeg_decode(encodedCb, q_inter) + refCb;
            encodedCr = jpeg_encode(diffCr, q_inter);
            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 + (nnz(encodedY) + nnz(encodedCb) + nnz(encodedCr)) * 2;
        end
    elseif frameType == 'B'
        prevIdx = find(strcmp(frameTypes(1:i-1), 'I') | strcmp(frameTypes(1:i-1), 'P'), 1, 'last');
        nextIdx = find(strcmp(frameTypes(i+1:end), 'I') | strcmp(frameTypes(i+1:end), 'P'), 1, 'first');
        if ~isempty(nextIdx)
            nextIdx = nextIdx + i;
        end
        if ~isempty(prevIdx) && ~isempty(nextIdx) && nextIdx <= numFrames
            refY = (double(decodedRGBFrames(:,:,1,prevIdx)) + double(decodedRGBFrames(:,:,1,nextIdx))) / 2;
            refCb = (double(decodedRGBFrames(:,:,2,prevIdx)) + double(decodedRGBFrames(:,:,2,nextIdx))) / 2;
            refCr = (double(decodedRGBFrames(:,:,3,prevIdx)) + double(decodedRGBFrames(:,:,3,nextIdx))) / 2;
            diffY = motion_compensate(Y, refY, blockSize, searchRange, skipThreshold);
            diffCb = motion_compensate(Cb, refCb, blockSize, searchRange, skipThreshold);
            diffCr = motion_compensate(Cr, refCr, blockSize, searchRange, skipThreshold);
            encodedY = jpeg_encode(diffY, q_inter); decodedY = jpeg_decode(encodedY, q_inter) + refY;
            encodedCb = jpeg_encode(diffCb, q_inter); decodedCb = jpeg_decode(encodedCb, q_inter) + refCb;
            encodedCr = jpeg_encode(diffCr, q_inter); 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', 'B', 'Y', encodedY, 'Cb', encodedCb, 'Cr', encodedCr, 'RefPrev', prevIdx,
'RefNext', nextIdx);
            encodedTotalBytes = encodedTotalBytes + (nnz(encodedY) + nnz(encodedCb) + nnz(encodedCr)) * 2;
        end
    end
end
vOrig = VideoWriter('original_video.avi');
open(vOrig)
for i = 1:numFrames
    writeVideo(vOrig, ycbcr2rgb(uint8(originalYCbCrFrames(:,:,:,i))));
end
close(vOrig)
vDec = VideoWriter('decoded_video.avi');
open(vDec)
for i = 1:numFrames
    writeVideo(vDec, decodedRGBFrames(:,:,:,i));
```

```
end
close(vDec)
rawSize = rows * cols * 3 * numFrames;
rawSizeKB = rawSize / 1024;
encodedSizeKB = encodedTotalBytes / 1024;
compressionRatio = rawSizeKB / encodedSizeKB;
mse = mean((double(decodedRGBFrames(:)) - double(uint8(originalYCbCrFrames(:)))) .^ 2);
psnrVal = 10 * log10(255^2 / mse);
fprintf('Raw Size: %.2f KB\n', rawSizeKB);
fprintf('Encoded Size: %.2f KB\n', encodedSizeKB);
fprintf('Compression Ratio: %.2f\n', compressionRatio);
fprintf('MSE: %.2f\n', mse);
fprintf('PSNR: %.2f dB\n', psnrVal);
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

%% --- JPEG Encode ---
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
            sub = block(i:i+blockSize-1, j:j+blockSize-1);
            dctB = dct2(sub);
            quant = round(dctB ./ Q);
            encodedBlock(i:i+blockSize-1, j:j+blockSize-1) = quant;
        end
    end
end

%% --- JPEG Decode ---
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
            quant = encodedBlock(i:i+blockSize-1, j:j+blockSize-1);
            dctB = quant .* Q;
            idctB = idct2(dctB);
            decodedBlock(i:i+blockSize-1, j:j+blockSize-1) = idctB;
        end
    end
end
```