EE 596 – MINI PROJECT

SEWWANDI E.A.T.

E/17/326

SEMESTER 7

24/07/2023

1. OBJECTIVES

1.1 To investigate the basic functions of an image coding system.

1.2 To investigate the basic functions of block-based video coding system.

1.3 To investigate the Rate – Distortion optimization techniques used in video coding.

2. INTRODUCTION

Image and video coding are critical processes for compressing and storing visual data in digital formats. These techniques are crucial for efficient image and video transmission, storage, and display in a variety of applications such as multimedia communication, broadcasting, video streaming, and digital archiving. Image coding is the process of reducing the amount of data needed to represent an image while still maintaining an acceptable level of visual quality.

Hybrid video coding is an advanced coding technique which is derived from both predictive coding and transform coding. Hybrid video coding framework is commonly used in modern video coding standards, e.g., H.26x, MPEG2/4, AVS, HEVC, etc. Objective of this project is to implement a simplified hybrid video codec with coding tools like discrete cosine transformation, quantization, prediction, and entropy coding.

JPEG is a popular image coding technique that compresses images using a combination of quantization, transform coding, and predictive coding. Images in the JPEG algorithm are typically divided into 8x8 macroblocks and transformed to the frequency domain using the Discrete Cosine Transform (DCT). The transformed coefficients are quantized, which reduces their compression precision. Entropy coding techniques, such as Huffman coding, are then used to efficiently encode the quantized coefficients. During decompression, the encoded data is reversed, allowing the quantized coefficients to be reconstructed. To retrieve the approximate pixel values, dequantization and inverse DCT are used, resulting in a compressed but visually similar representation of the original image.

Motion compensation is a video compression technique that reduces the effect of motion between consecutive frames or images. In motion estimation, differences between consecutive frames are analyzed to estimate motion vectors, which represent the displacement of objects or regions between frames. For motion estimation, various techniques such as block matching, optical flow, and feature-based methods can be used. After determining the motion vectors, the motion compensation step involves reconstructing the current frame by shifting and blending pixels from the reference frame based on the estimated motion vectors.

3. METHODOLOGY

3.1 IMAGE COMPRESSION

The "Lenna" image is a famous digital image widely used in the field of image processing and computer vision. For the image compression part, the "Lenna" image was selected. It is a colored image, but for the sake of simplicity, color image was converted to grayscale image and the required tasks were carried out. If it is necessary to perform this process for a color image, it is first required to convert it to the YC_bC_r color space. Subsequently, each of the three layers must be compressed separately, and finally, they are combined after compression.

Figure 1: Original colored image and grayscale image

Then the image compression was performed using the procedure indicated below.

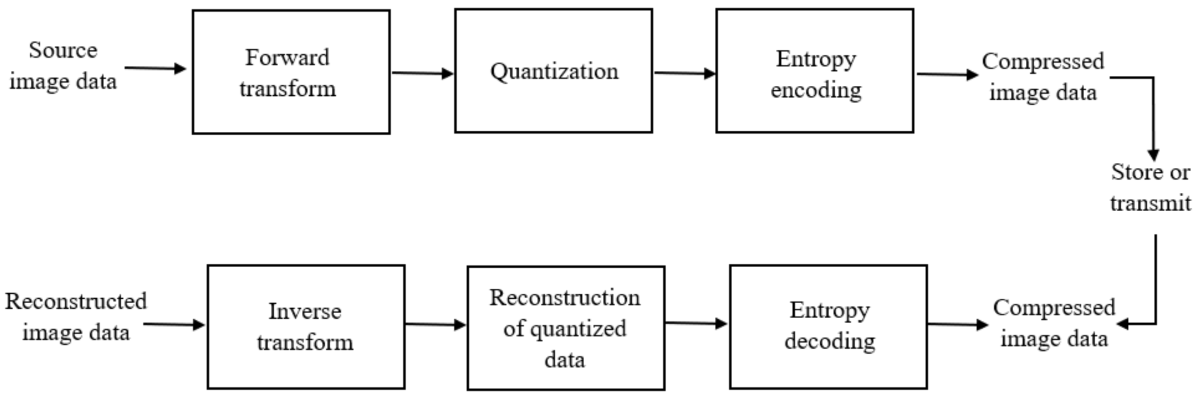


Figure 2: Basic image compression system

Initially, the grayscale image was divided into 8x8 macroblocks. Then the pixel values in the current macroblock are converted to double and centered by subtracting 128. This step is performed to ensure that the transform coefficients are centered around zero.

Forward Transform (Discrete Cosine Transform)

The Discrete Cosine Transform (DCT) is a widely used mathematical transformation technique that converts a signal or image from the spatial domain into the frequency domain. The DCT operates on a finite sequence of data points and produces a set of coefficients representing the original data in terms of its frequency components. It is closely related to the Fourier Transform, but whereas the Fourier Transform uses complex exponentials (sine and cosine functions) to represent the data, the DCT uses only real cosine functions. The general DCT equation for a 2D image is defined by the following equation.

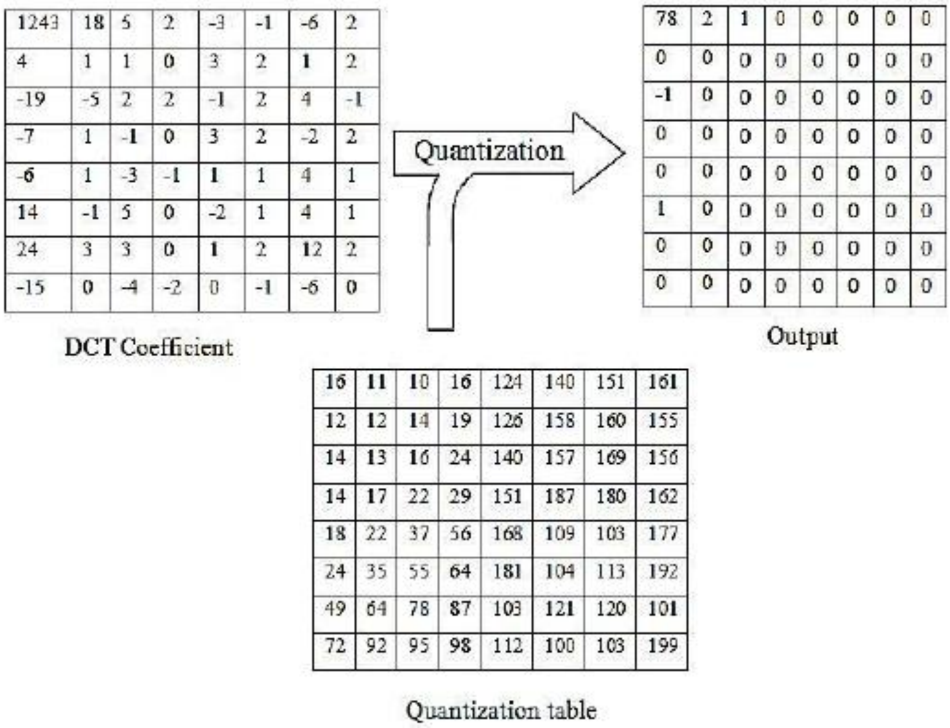
$$F(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\frac{(2x+1)u\pi}{2N}\cos\frac{(2y+1)v\pi}{2N}$$

Quantization

Quantization is an important step in lossy image compression algorithms like JPEG that reduces file size while maintaining visual quality. To quantize the DCT coefficients of the image, simply divide each coefficient by the corresponding value in the quantization matrix and round the result to the nearest integer. The higher the value in the quantization matrix, the more the corresponding coefficient will be reduced, resulting in greater compression. JPEG standard quantization table was used as the quantization matrix. The matrix multiplication factor was used to get low, medium and high quality outputs. The multiplication factor is inversely proportional to the quality of the output image.

```
% Multiplication factor
mf = 1;
% Standard quantization matrix used in JPEG compression
base_matrix = [16 11 10 16 24 40 51 61;
       12 12 14 19 26 58 60 55;
       14 13 16 24 40 57 69 56;
       14 17 22 29 51 87 80 62;
       18 22 37 56 68 109 103 77;
       24 35 55 64 81 104 113 92;
       49 64 78 87 103 121 120 101;
       72 92 95 98 112 100 103 99];
% Quantization matrix
quant = mf * base_matrix;
```

A common quantization process behaves like following.



DCT Coefficient

1243	18	5	2	-3	-1	-6	2
4	1	1	0	3	2	1	2
-19	-5	2	2	-1	2	4	-1
-7	1	-1	0	3	2	-2	2
-6	1	-3	-1	1	1	4	1
14	-1	5	0	-2	1	4	1
24	3	3	0	1	2	12	2
-15	0	-4	-2	0	-1	-6	0

Output

78	2	1	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantization table

16	11	10	16	124	140	151	161
12	12	14	19	126	158	160	155
14	13	16	24	140	157	169	156
14	17	22	29	151	187	180	162
18	22	37	56	168	109	103	177
24	35	55	64	181	104	113	192
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	199

Figure 3: Quantization of DCT transformed image matrix

i. Low quality quantization (Quantization matrix multiplication factor = 10)

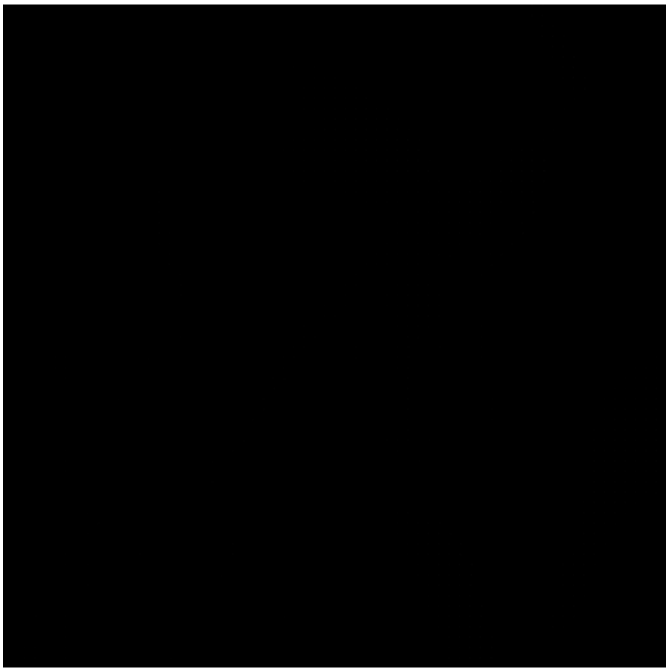


Figure 4: Low quality quantized image

ii. Medium quality quantization (Quantization matrix multiplication factor = 1)

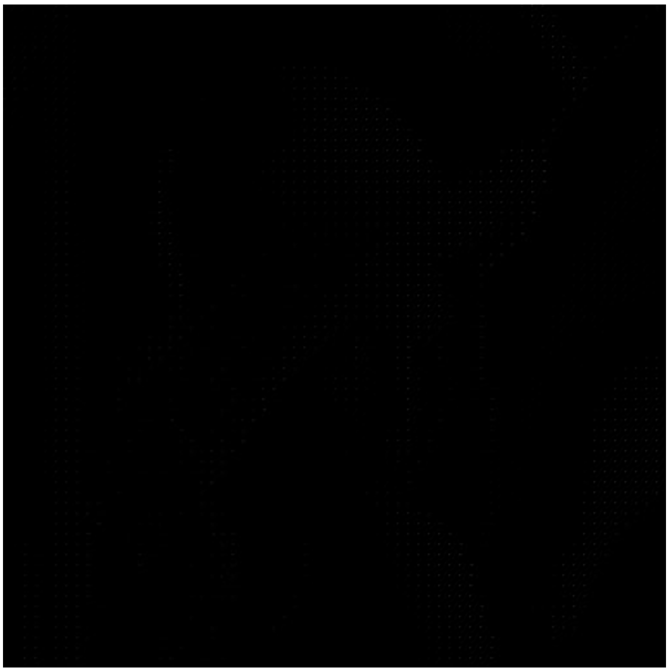


Figure 5: Medium quality quantized image

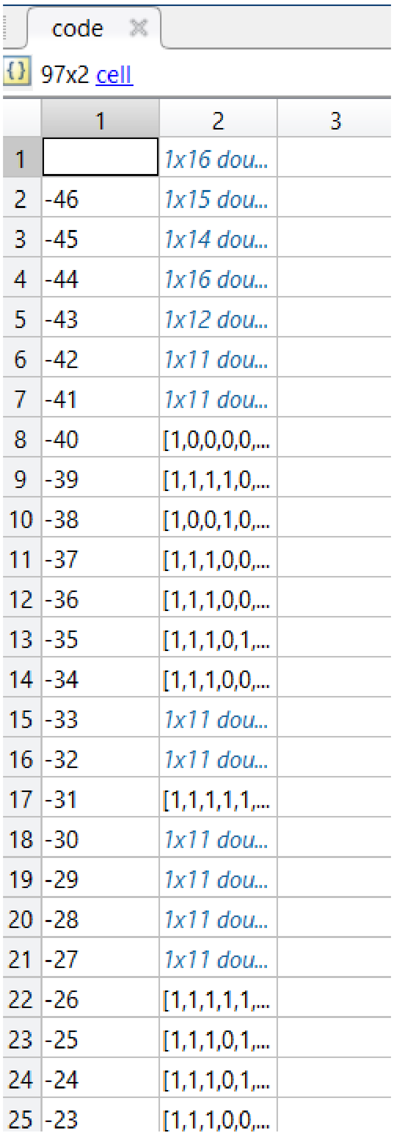
iii. High quality quantization (Quantization matrix multiplication factor = 0.1)

Figure 6: High quality quantized image

Entropy Encoding (Huffman Encoding)

Huffman encoding is a lossless data compression algorithm that uses frequency of occurrence to assign variable-length codes to symbols. The goal of Huffman encoding is to reduce overall data size by representing frequently occurring symbols with shorter codes and less frequently occurring symbols with longer codes. As a result, the original data is represented more efficiently.

After the quantization step, entropy encoding was performed using the Huffman encoding algorithm. The Huffman codebook was generated using 'huffmandict' function with intensity values and probabilities of the quantized image as the inputs. Subsequently, 'huffmanenco' function was used to encode the quantized image using the generated codebook. The resulting encoded bit array was then saved to a text file, providing a compressed and efficient representation of the original image.


code
97x2 cell

	1	2	3
1		1x16 dou...	
2	-46	1x15 dou...	
3	-45	1x14 dou...	
4	-44	1x16 dou...	
5	-43	1x12 dou...	
6	-42	1x11 dou...	
7	-41	1x11 dou...	
8	-40	[1,0,0,0,0,...	
9	-39	[1,1,1,1,0,...	
10	-38	[1,0,0,1,0,...	
11	-37	[1,1,1,0,0,...	
12	-36	[1,1,1,0,0,...	
13	-35	[1,1,1,0,1,...	
14	-34	[1,1,1,0,0,...	
15	-33	1x11 dou...	
16	-32	1x11 dou...	
17	-31	[1,1,1,1,1,...	
18	-30	1x11 dou...	
19	-29	1x11 dou...	
20	-28	1x11 dou...	
21	-27	1x11 dou...	
22	-26	[1,1,1,1,1,...	
23	-25	[1,1,1,0,1,...	
24	-24	[1,1,1,0,1,...	
25	-23	[1,1,1,0,0,...	

Figure 7: Haffman codebook

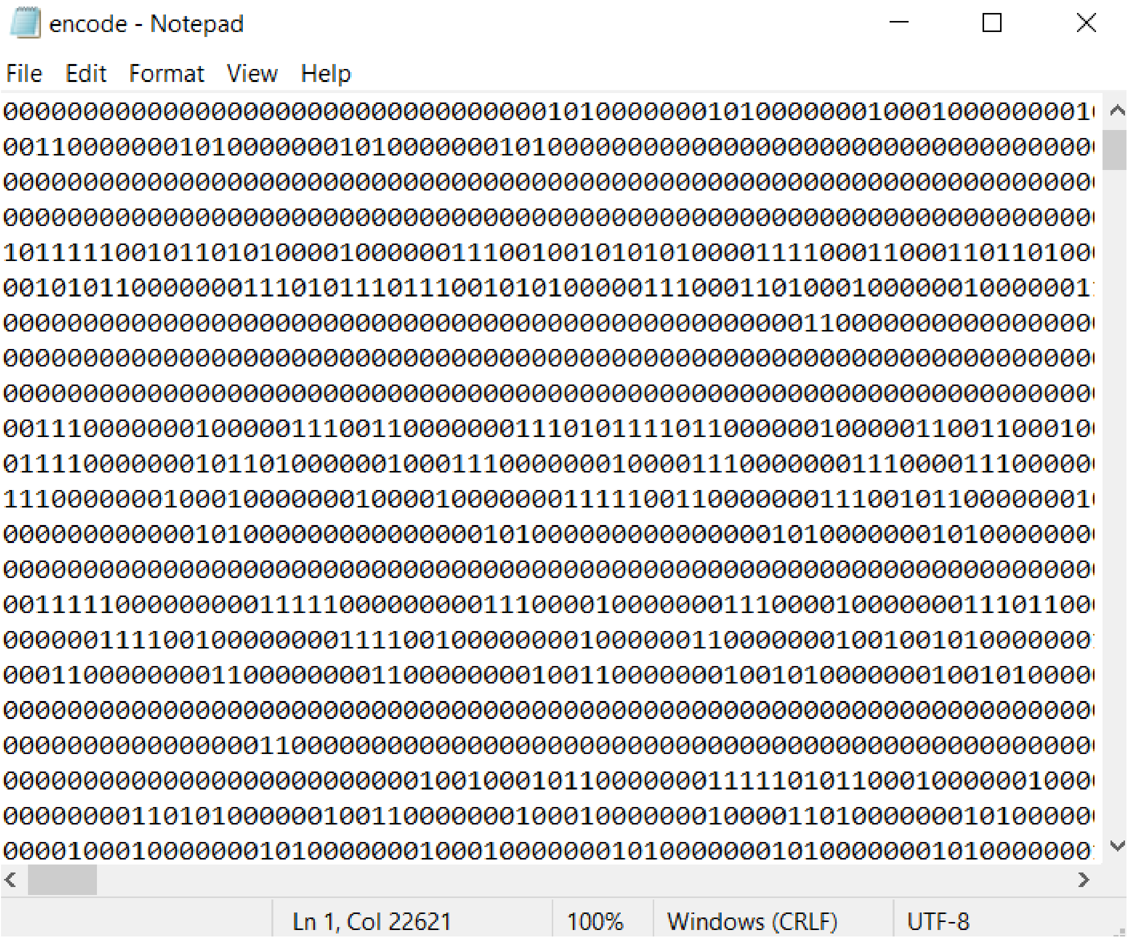


Figure 8: Haffman encoded bit sream

Entropy Decoding (Huffman Decoding)

For decoding process, initially the text file was read, and then encoded bits were converted to a numerical array. Using 'huffmandeco' function with the Huffman codebook, the image was decoded. The decoded image was reshaped to its original size, recovering the original data.

Dequantization

In the dequantization step, the dequantized coefficients were obtained by multiplying each quantized coefficient by the corresponding element in the quantization matrix. This process helps to recover the original coefficient values that were quantized during the compression stage.

Inverse Transform (Inverse Discrete Cosine Transform)

The Inverse Discrete Cosine Transform (IDCT) was applied to each spatial component of the dequantized coefficients to compute the pixel values. After applying the inverse DCT, the resulting pixel values were centered by adding 128. This step assists in returning the pixel values to their original range.

After completing all of these steps, the reconstructed image was obtained. As previously stated, quality can be adjusted by elementwise multiplying the quantization matrix. low, medium, and high quality outputs were obtained by varying the multiplication factor. The multiplication factor is inversely proportional to the resolution.

i. Low quality output (Quantization matrix multiplication factor = 10)

Quantization matrix multiplication factor	= 10
Entropy of the output image	= 5.3405
PSNR value	= 27.3295

Figure 9: Low quality output image

ii. Medium quality output (Quantization matrix multiplication factor = 1)

Quantization matrix multiplication factor	= 1
Entropy of the output image	= 7.4360
PSNR value	= 35.8081

Figure 10: Medium quality output image

iii. High quality output (Quantization matrix multiplication factor = 0.1)

Quantization matrix multiplication factor	= 0.1
Entropy of the output image	= 7.4474
PSNR value	= 43.8421

Figure 11: High quality output image

The image encoder's compression ratio was adjusted to meet a target bit rate of 626 kbps while maximizing the PSNR value.

3.2 VIDEO COMPRESSION

Video compression is a vital technique used to reduce the size of digital video files while maintaining an acceptable level of quality. It enables us to reduce the amount of bandwidth needed to transfer videos, providing a better user experience when media is accessed. Two primary compression types are employed: lossy and lossless. Lossy compression achieves higher compression ratios by discarding less noticeable visual information. Lossless compression retains all original data without quality loss, but results in larger file sizes. Video compression relies on algorithms like transform coding, motion compensation, and entropy encoding. Transform coding converts frames into a frequency domain representation (e.g., DCT) to remove spatial redundancies. Motion compensation analyzes frame differences, encoding only motion information. Entropy encoding (e.g., Huffman, Arithmetic, Run length coding) compresses data by efficiently representing common patterns. Popular video compression standards include H.264/AVC, H.265/HEVC, and VP9. The process used in the H.264 hybrid video codec is shown below.

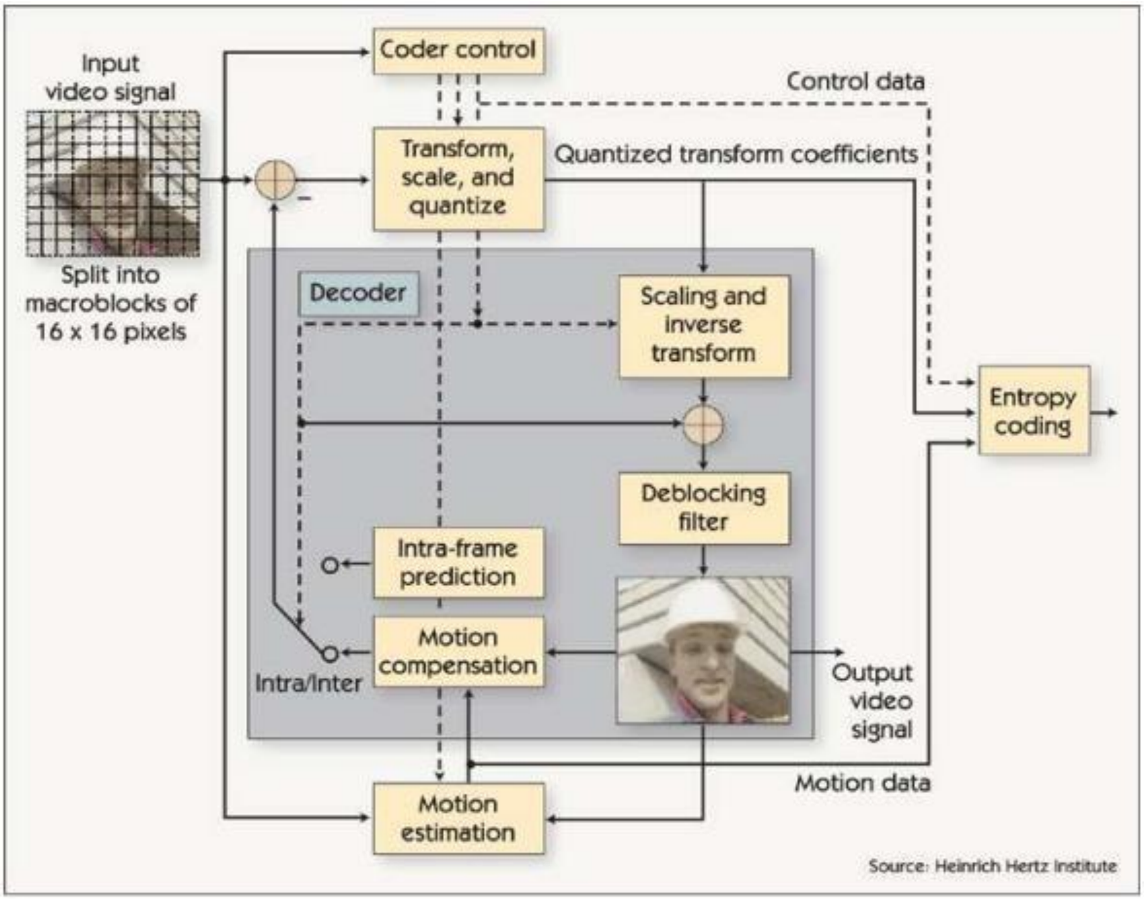


Figure 12: Block diagram of H.264 hybrid video codec

I have used the similar method for the video compression. First, I chose a video with a duration of 20 seconds, a resolution of 1920x1080, and a frame rate of 30 frames per second. I chose 1s (30 frames) from that for this activity. Due to the limitations of my processing power, I converted the color images extracted from the original video to grayscale images and reduced the frame size to 1280x720.

Figure 13: First frame of the original video (1920x1080)

Figure 14: First frame of the original video (1920x1080)

The padding technique is used to ensure that frames can be divided into 8x8 macroblocks without any remainder. Then, the padded images were divided into 8x8 macroblocks.

For video compression IPPP… structure was used. In video compression using the IPPP structure, the first frame was compressed without utilizing motion vectors. Instead, the same method used in image compression was applied to compress the first frame. This typically involves techniques such as intra-frame coding, where spatial redundancies within the frame are exploited.

The reconstructed image obtained from the compression of the first frame was then taken as the reference frame for the second frame. The reference frame served as the basis for motion estimation and compensation in subsequent frames. For the second frame and onwards, motion vectors were employed to describe the displacement of macroblocks or regions between the current frame and the reference frame.

Figure 15: Reconstructed I-frame

Motion Estimation

Motion estimation is the process of analyzing consecutive video frames to estimate the motion between them. It involves dividing frames into macroblocks, finding the best match in a reference frame, and calculating motion vectors. The block matching technique was used to estimate the motion vectors. In the block matching technique, first each image is subdivided into 8x8 macroblocks and finding a motion vector for each block. A search should be performed within a predefined range to find the best match that minimizes an error measure. To find the best match in block matching, various search algorithms can be used, such as full/sequential search, logarithmic search, and hierarchical search. Full search exhaustively compares all candidate blocks, while logarithmic search reduces complexity by dividing the search range. Hierarchical search performs motion estimation at different resolutions. During this project, sequential search technique was used. The error measure can be based on metrics like the sum of absolute differences (SAD) or mean squared error (MSE) between corresponding blocks in the current and reference frames. The SAD matrix was used for that purpose. The motion vector represents the motion between the macroblocks, indicating the direction and magnitude of the movement.

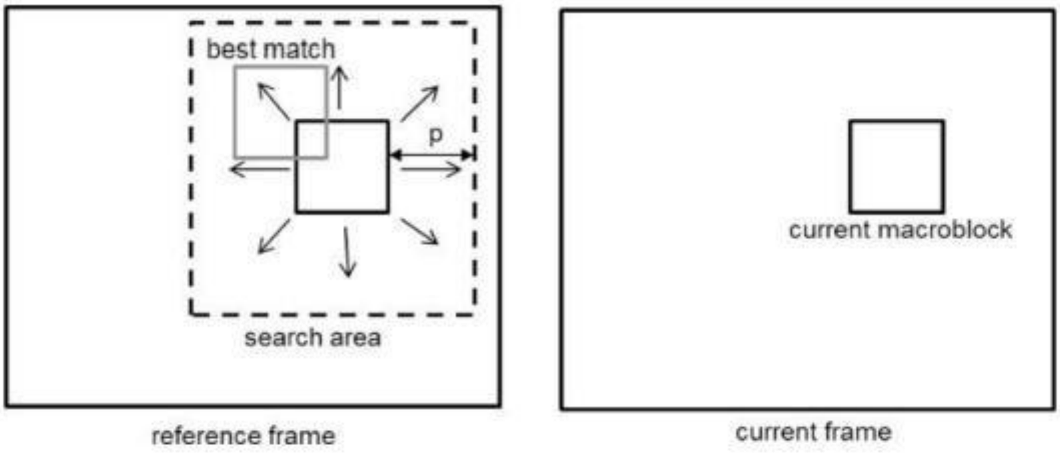


Figure 16: Sequential search of the macroblock

motion_vec | motion_vec{1, 1}

motion_vec{1, 1}

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1		[2,2]	[2,3]	[2,4]	[2,5]	[2,6]	[1,7]	[2,10]	[2,10]	[2,11]	[2,11]	[2,13]	[2,14]	[2,15]	[2,16]	[2,17]	[2,18]	[2,18]
2	[2,2]	[2,2]	[2,3]	[2,4]	[2,5]	[2,6]	[2,7]	[2,10]	[3,10]	[3,11]	[2,11]	[3,13]	[2,15]	[3,15]	[3,16]	[3,17]	[3,18]	[3,19]
3	[3,2]	[3,2]	[3,3]	[3,4]	[3,5]	[3,6]	[3,7]	[3,10]	[3,10]	[3,10]	[3,11]	[4,13]	[4,14]	[4,15]	[4,16]	[4,17]	[4,18]	[4,18]
4	[4,2]	[4,2]	[4,3]	[4,4]	[4,5]	[4,6]	[4,7]	[4,10]	[4,10]	[4,10]	[5,12]	[4,14]	[5,14]	[5,15]	[4,17]	[4,17]	[4,18]	[4,18]
5	[5,2]	[5,2]	[5,3]	[5,4]	[5,5]	[5,6]	[5,7]	[5,9]	[5,10]	[6,11]	[5,13]	[5,14]	[6,14]	[6,15]	[5,15]	[5,16]	[5,17]	[5,18]
6	[6,2]	[6,2]	[6,3]	[6,4]	[6,5]	[6,6]	[7,8]	[6,10]	[6,10]	[6,10]	[6,13]	[6,14]	[6,14]	[6,14]	[6,15]	[6,16]	[6,17]	[6,18]
7	[7,2]	[7,2]	[7,3]	[8,5]	[7,5]	[7,6]	[8,8]	[8,9]	[8,9]	[7,10]	[7,13]	[7,13]	[7,14]	[7,14]	[8,16]	[8,16]	[7,17]	[7,20]
8	[8,2]	[8,2]	[9,4]	[8,4]	[8,5]	[8,6]	[8,9]	[8,9]	[8,9]	[8,12]	[9,13]	[9,13]	[9,13]	[9,15]	[10,16]	[8,16]	[9,17]	[8,20]
9	[9,2]	[9,2]	[9,4]	[9,4]	[9,5]	[10,7]	[9,9]	[10,9]	[9,9]	[9,12]	[10,12]	[10,13]	[10,13]	[10,15]	[10,16]	[9,17]	[9,19]	[9,20]
10	[10,2]	[10,2]	[10,4]	[10,4]	[10,5]	[10,8]	[10,8]	[11,9]	[11,10]	[10,12]	[11,12]	[11,13]	[10,13]	[10,14]	[10,15]	[11,18]	[11,19]	[11,19]
11	[11,2]	[11,2]	[11,3]	[11,4]	[11,7]	[12,7]	[13,8]	[13,8]	[12,10]	[11,12]	[12,12]	[11,13]	[11,13]	[11,15]	[11,15]	[12,17]	[12,18]	[12,20]
12	[12,2]	[12,2]	[12,3]	[12,4]	[12,7]	[13,7]	[13,8]	[13,8]	[13,10]	[13,11]	[13,12]	[12,13]	[12,13]	[12,16]	[13,17]	[13,18]	[12,19]	[13,20]
13	[13,2]	[14,3]	[15,3]	[15,5]	[14,7]	[15,6]	[15,7]	[14,8]	[14,9]	[14,11]	[14,12]	[13,13]	[13,13]	[13,16]	[13,17]	[13,18]	[13,19]	[13,20]
14	[14,3]	[15,3]	[15,5]	[15,6]	[15,7]	[15,7]	[16,7]	[15,8]	[15,9]	[16,11]	[15,12]	[14,12]	[15,14]	[14,16]	[14,17]	[15,18]	[14,19]	[14,19]
15	[16,2]	[16,4]	[15,5]	[16,5]	[17,6]	[16,7]	[15,8]	[15,8]	[15,9]	[17,11]	[16,12]	[15,12]	[15,13]	[15,15]	[16,17]	[16,17]	[15,19]	[15,19]
16	[16,2]	[17,3]	[16,5]	[17,5]	[17,6]	[17,6]	[16,7]	[16,8]	[16,11]	[18,11]	[18,11]	[16,12]	[16,13]	[16,15]	[16,17]	[18,18]	[17,18]	[18,19]
17	[17,3]	[17,4]	[17,5]	[18,6]	[18,6]	[18,6]	[17,7]	[17,8]	[18,11]	[18,11]	[19,11]	[17,12]	[17,14]	[17,16]	[17,17]	[18,17]	[19,19]	[19,18]
18	[18,3]	[18,4]	[18,4]	[18,5]	[18,6]	[18,6]	[18,7]	[18,8]	[18,11]	[19,11]	[19,11]	[18,12]	[18,15]	[18,16]	[18,17]	[20,17]	[19,18]	[19,18]
19	[20,2]	[19,4]	[19,4]	[19,6]	[19,6]	[20,7]	[20,8]	[19,8]	[19,11]	[20,11]	[19,11]	[20,12]	[20,14]	[20,15]	[20,16]	[20,18]	[20,18]	[20,18]
20	[21,2]	[22,4]	[20,5]	[20,6]	[20,6]	[20,7]	[20,7]	[20,9]	[21,10]	[20,11]	[20,11]	[21,13]	[21,14]	[21,15]	[21,16]	[21,17]	[21,18]	[21,18]
21	[22,2]	[22,3]	[21,5]	[22,5]	[22,5]	[22,6]	[21,7]	[21,10]	[21,10]	[21,11]	[21,11]	[21,14]	[21,15]	[22,15]	[22,16]	[22,17]	[22,18]	[22,18]
22	[22,3]	[22,3]	[22,5]	[22,5]	[23,6]	[23,7]	[23,8]	[24,8]	[22,10]	[22,10]	[22,11]	[22,14]	[22,15]	[23,15]	[23,16]	[23,17]	[23,18]	[23,18]
23	[23,3]	[23,2]	[23,5]	[23,6]	[24,6]	[25,6]	[24,7]	[24,8]	[23,9]	[23,10]	[25,13]	[23,14]	[23,15]	[23,16]	[24,16]	[24,17]	[24,17]	[23,18]
24	[24,2]	[24,4]	[24,4]	[25,5]	[25,6]	[24,6]	[24,7]	[24,8]	[24,9]	[24,10]	[24,13]	[24,14]	[24,15]	[24,16]	[24,17]	[24,17]	[24,17]	[24,18]
25	[25,2]	[25,4]	[25,5]	[26,5]	[25,6]	[26,7]	[25,7]	[25,8]	[25,9]	[25,10]	[25,13]	[25,13]	[25,15]	[25,16]	[25,16]	[25,17]	[25,17]	[25,18]

Figure 17: Motion vectors of frame number 2

Motion Compensation

Motion compensation is a technique used in video coding and compression to reduce temporal redundancy between consecutive frames. In motion compensation, a reference frame is selected as a prediction for the current frame. The motion vectors obtained from motion estimation techniques, such as block matching, are used to shift or transform the reference frame to align it with the current frame. This creates a predicted frame that represents the motion information.

Residual Calculation

The difference between the predicted and actual current frames was calculated and encoded. The run length encoding technique was used for the encoding. After quantizing the residual image, the discrete cosine transform was applied to all of the macroblocks. Because there were many zeros in the residual image after applying the discrete cosine transform, run length coding was used for the transmission part. Motion compensation achieves compression by transmitting only the motion vectors and residual information, rather than the entire frame, by taking advantage of the fact that consecutive frames in a video sequence frequently have similarities and exhibit motion patterns. The residual can be calculated using the equation below.

Residual = Current Frame – Predicted Frame

Figure 18: Predicted frame number 2

Figure 19: Residual of frame number 2

During video decoding, the array was first decoded using run length decoding. The first frame, known as the I-frame, was decoded independently using intra coding. The decoded residual was obtained through inverse discrete cosine transformation and dequantization. Using motion vectors, the predicted frame was constructed and combined with the decoded residual to reconstruct subsequent frames, known as P-frames (inter-coded frames). Reconstructed current frame can be calculated by the following equation.

Decoded Residual + Predicted Frame = Reconstructed Current Frame

Figure 20: Reconstructed frame number 2

3.3 IMPROVED HYBRID VIDEO CODEC

Rate-distortion optimization in video coding balances video quality and compression efficiency by choosing optimal coding parameters and techniques. It minimizes distortion at a given bit rate or minimizes the bit rate for a desired level of distortion. Choosing appropriate quantization parameters, modes for intra- and inter-frame coding, and rate control algorithms are all part of this process. The selection of quantization parameters governs the trade-off between compression and quality, whereas mode selection optimizes coding techniques for different frames. Rate control algorithms allocate bits dynamically to achieve a desired bit rate. Rate-distortion optimization algorithms explore the trade-off space and find the best operating point using mathematical models and optimization techniques. Video codecs can achieve efficient performance by employing rate-distortion optimization.

Distortion varies with the frame transmission rate. Distortion is inverse of the quality of reconstructed matrix. Also, PSNR measuring the quality of the image. Therefore, here distortion is considered as the inverse of the PSNR value. For this rate distortion optimization task JPEG quantization matrix was multiplied by different scaling factors and measured the PSNR values and the reconstructed image sizes. Assume that one frame will be transmitted per second. Then the reconstructed image size and the transmission rate is equal.

Table 1: Variation of distortion vs transmission rate for different quantization matrix multiplication factors

Quantization Matrix Multiplication Factor	Transmission Rate / KBps	PSNR	Distortion = 1/PSNR
0.1	32	43.8421	0.022809126
0.2	30	40.8108	0.024503318
0.3	40	39.4472	0.025350342
0.4	36	38.5435	0.025944712
0.5	32	37.8647	0.026409822
0.6	30	37.3252	0.026791551
0.7	28	36.8581	0.027131078
0.8	30	36.4605	0.027426941
0.9	29	36.1131	0.027690783
1	28	35.8081	0.027926642
2	23	33.7042	0.029669893
3	18	32.3327	0.030928441
4	18	31.2578	0.031992015
5	15	30.4108	0.032883055
6	14	29.631	0.033748439
7	12	28.9939	0.034490013
8	13	28.4568	0.035140986
9	12	27.7852	0.035990383
10	11	27.3295	0.036590497
20	8	23.8962	0.041847658
30	6	21.1506	0.047279983
40	6	19.8085	0.050483378
50	5	17.9576	0.055686729

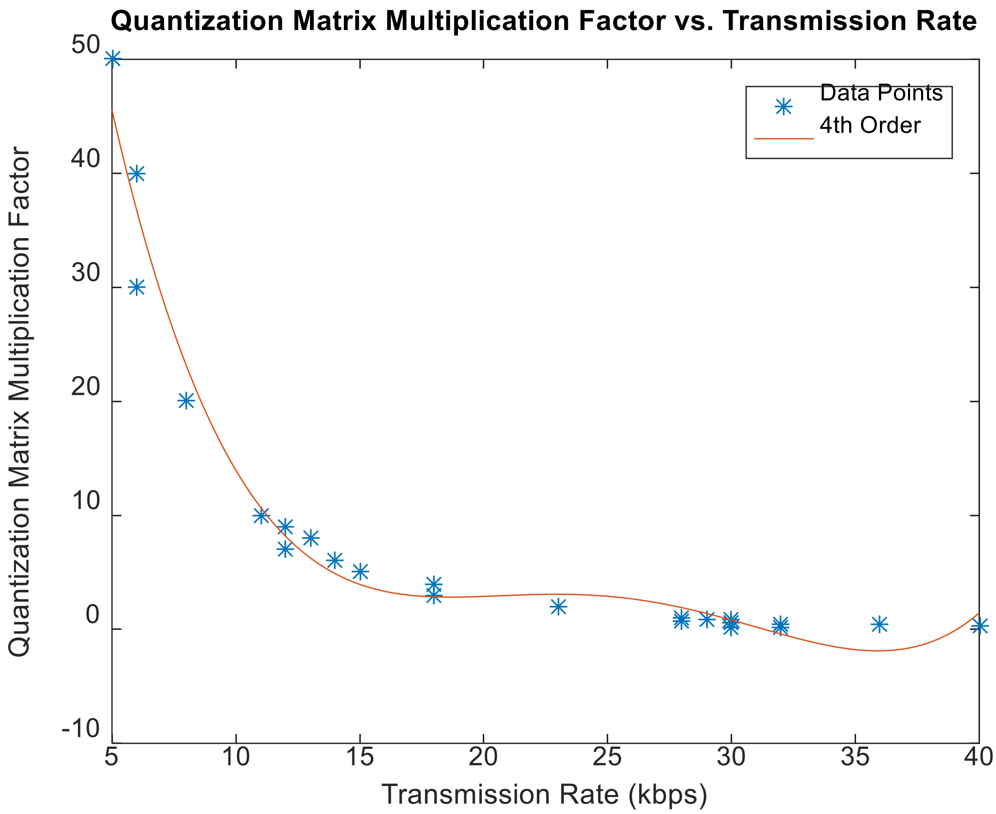


Figure 21: Variation of the quantization matrix multiplication factor vs transmission rate

Figure 21 shows that the quantization matrix multiplication factor and transmission rate have a 4th order polynomial relationship. The general formula for a polynomial of fourth order is shown below.

$$y = c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

y = Quantization matrix multiplication factor

x = Transmission rate

Coefficients:

$$c_4 = 0.0003$$

$$c_3 = -0.0327$$

$$c_2 = 1.2140$$

$$c_1 = -19.4093$$

$$c_0 = 116.1401$$

We can find the quantization matrix multiplication factor by substituting the target transmission rate using that equation.

REFERENCES

1. asecuritysite.com. (n.d.). DCT. [online] Available at: https://asecuritysite.com/comms/dct [Accessed 20 June. 2023].

2. Song, M.-S. (n.d.). Entropy Encoding in Wavelet Image Compression. [online] Available at: https://www.siue.edu/~msong/Research/entropy.pdf [Accessed 22 June. 2023].

3. www.dcs.warwick.ac.uk. (n.d.). Block-Matching Motion Compensation. [online] Available at: https://www.dcs.warwick.ac.uk/research/mcg/bmmc/index.html [Accessed 24 June. 2023].

4. Wang, H. and Kwong, S. (2008). Rate-Distortion Optimization of Rate Control for H.264 With Adaptive Initial Quantization Parameter Determination. IEEE Transactions on Circuits and Systems for Video Technology, [online] 18(1), pp.140–144. doi:https://doi.org/10.1109/TCSVT.2007.913757 [Accessed 24 June. 2023].

APPENDIX

Image Coding

```
%%% Image Coding %%%

clear all;
clc;
%% Source image data

original_image = imread('Lenna_(test_image).png');
% figure,
% imshow(original_image);
% title('Original Image');

gray_image = rgb2gray(original_image);
% figure
% imshow(gray_image);
% title('Gray Image');

% The size of the grayscale image
[h, w] = size(gray_image);
r = h/8;
c = w/8;

% Set the target bit rate (in kbps)
% E/17/326 -> 326 + 300 = 626 kbps
target_bit_rate = 626;

%% Apply DCT & quantization

% Set the multiplication factor to achieve the desired bit
rate
% mf = (target_bit_rate * 1000) / (8 * numel(gray_image));

% Multiplication factor
mf = 1;
% Standard quantization matrix used in JPEG compression
base_matrix = [16 11 10 16 24 40 51 61;
       12 12 14 19 26 58 60 55;
       14 13 16 24 40 57 69 56;
       14 17 22 29 51 87 80 62;
       18 22 37 56 68 109 103 77;
       24 35 55 64 81 104 113 92;
       49 64 78 87 103 121 120 101;
       72 92 95 98 112 100 103 99];
% Quantization matrix
quant = mf * base_matrix;
```

```
% Call the quantized_dct function
Q = quantized_dct(gray_image,quant);

quantized_image = uint8(Q);
% figure;
% imshow(quantized_image);
%title('Quantized Image');

%% Entropy encoding

% The intensity values of the compressed image
[g,~,intensity_val] = grp2idx(Q(:));
% The frequency of occurrence for each intensity value
frequency = accumarray(g,1);
% The probabilities of each intensity value
probability = frequency./(h*w);
% T = table(intensity_val,frequency,probability);
% Create the Huffman codebook
code = huffmandict(intensity_val,probability);
% Encode the compressed image
encode = huffmanenco(Q(:),code);

% Calculate the actual bit rate
actual_bit_rate = numel(encode) / numel(gray_image) * 8;

% Calculate the compression ratio
compression_ratio = (h * w * 8) / numel(encode);

%% Store compressed image data

% The encoded image is saved to a text file
text1 = 'encode.txt';
% Read the text file and the encoded bits are converted to a
numerical array
fid = fopen(text1,'w');
fprintf(fid,num2str(encode));
fclose(fid);

f = fopen(text1); % Open the text file
data = textscan(f,'%s');% Read the contents of the text file
fclose(f);% Close the text file
bits = char(data{:});% Convert the cell array of strings into
a character array

test = [];
len = size(encode);
for q = 1:len(1)
    test(q) = bits(q)-48; % Convert character into numerical
value by subtracting 48 from ASCII value
end
```

```
%% Entropy decoding

% The numerical array is passed to 'huffmandeco'along with the
Huffman codebook to decode the image
decode = huffmandeco(test',code);
% The decoded image is reshaped
image1 = reshape(decode,h,w);

%% Apply dequantization & IDCT

% Call the dequantized_idct function
image2 = dequantized_idct(image1,quant);

%% Reconstructed image data

reconstructed_image = uint8(image2);

% Calculate PSNR value
psnr_val = psnr(reconstructed_image, gray_image);

% figure;
% imshow(reconstructed_image);
%title('Output Image');
imwrite(gray_image, 'input.jpg');
imwrite(quantized_image, 'quantized image.jpg');
imwrite(uint8(image2), 'output.jpg');




function Q = quantized_dct(gray_image,quant_mat)

    % The size of the grayscale image
    [h, w] = size(gray_image);
    r = h/8;
    c = w/8;
    s = 1;

    for i=1:r
        e = 1;
        for j=1:c

            % Divide grayscale image into 8x8 macroblocks
            macroblock = gray_image(s:s+7,e:e+7);

            % Convert the pixel values in the current
macroblocks to double and centre them by subtracting
128(Halfway between the minimum and maximum possible pixel
values in an 8-bit grayscale image)
            centre = double(macroblock) - 128;
```

```matlab
            % DCT(Discrete Cosine Transform)
            dc = dct2(centre);

            % Quantization
            Q(s:s+7, e:e+7) = round(dc ./ quant_mat);

            e = e + 8;
        end
        s = s + 8;
    end

end




function Out = dequantized_idct(constructed_image,quant_mat)

    % The size of the input constructed image
    [h, w] = size(constructed_image);
    r = h/8;
    c = w/8;
    s = 1;

    for i=1:r
        e = 1;
        for j=1:c
            qc = constructed_image(s:s+7, e:e+7);

            % Dequantization
            DQ = quant_mat .* qc;

            % Inverse DCT(Inverse Discrete Cosine Transform)
            idc = idct2(DQ);

            % Output matrix
            Out(s:s+7, e:e+7) = idc + 128;

            e = e + 8;
        end
        s = s + 8;
    end

end
```

Video Coding

```
%%% Video Coding %%%

clear all;
clc;

%% Read the original video file and extract 30 frames (1s
duration) from the video

video = VideoReader('squirrel.mp4');    % Read the video file
num_frames = 30;    % Number of frames to extract

% Read and save the specified number of frames from the video
for frame_count = 1:num_frames
     frame_file = strcat('frame',num2str(frame_count),'.jpg');
% Save the frame as a JPG image
     frame = read(video, frame_count);   % Read the next frame
     gray_frame = rgb2gray(frame);  % Convert the frame to
grayscale
     rs_frame = imresize(gray_frame, [720,1280]); % Resize the
image
     imwrite(rs_frame, frame_file); % Save frames
end

% Frame height and width
[rows , cols] = size(rs_frame);

% Save frame names
for i = 1 : num_frames
   frame_names{1,i} = strcat('frame',num2str(i),'.jpg');
end

% Save frame matrices
for j = 1 : num_frames
   frames{1,j} = imread(frame_names{1,j});
end

%% Rewrite the original video using extracted frames

original_vid = VideoWriter('original.mp4');
original_vid.FrameRate = video.FrameRate;
open(original_vid);
for j = 1:num_frames
    frame_count = uint8(frames{1,j});
    writeVideo(original_vid,frame_count);
end
% Close the original video file
close(original_vid);
```

```matlab
%% Padding

% The padding technique is used to ensure that frames can be
divided into 8x8 macroblocks without any remainder
for p = 1 : num_frames
    padd_frames{1,p} = padding(frames{1,p},8);
end

[r,c] = size(padd_frames{1,1});

%% Divide padded image into 8x8 macroblocks

for num = 1:num_frames
    macro{1,num} =
divide_into_macroblocks(padd_frames{1,num});
end

%% Motion estimation -> Motion vector calculation

[predict, motion_vec] = motion_vectors(macro);

%% Motion compensation -> Predicted frames

for i = 1:num_frames-1
    predict_img = combine_macroblocks(predict{1,i} ,rows ,
cols );
    predicted_frames{1,i}= predict_img;
%   frame_file =
strcat('predicted_frame',num2str(i+1),'.jpg');
%   imwrite(predict_img, frame_file);
end


%% Quantization

% Standard quantization matrix used in JPEG compression
quant = [16 11 10 16 24 40 51 61;
       12 12 14 19 26 58 60 55;
       14 13 16 24 40 57 69 56;
       14 17 22 29 51 87 80 62;
       18 22 37 56 68 109 103 77;
       24 35 55 64 81 104 113 92;
       49 64 78 87 103 121 120 101;
       72 92 95 98 112 100 103 99];

%% Apply image compression algorithm to referance frame (I
frame)

ref_frame = frames{1,1};
Q = quantized_dct(ref_frame,quant); % Call quantized_dct
function
```

```
quan{1,1} = Q;
I_frame = dequantized_idct(Q,quant);    % Call
dequantized_idct function
reconstructed_img{1,1} = I_frame;


%% Apply run length encoding for I frame

matrix = quan{1,1};
B = matrix';
C = reshape(B,1,[]);
D = runlength_encode(C);    % Call runlength_encode function
rl_array_I{1,1} = D;

%% Residual calculation -> Current macroblock - Predicted
macroblock

for i = 1:num_frames-1
    residual_img{1,i} = double(frames{1,i+1}) -
double(predicted_frames{1,i});
%   frame_file = strcat('residual', num2str(i+1), '.jpg');
%   imwrite(uint8(residual_img), frame_file);
end

%% Apply DCT and quantization for each residual image

for i = 1: num_frames-1
    Q = quantized_dct(residual_img{1,i},quant);  % Call
quantized_dct function
    encoded{1,i} = Q;
end

%% Apply run length encoding for each residual image

for i = 1:num_frames-1
   matrix = encoded{1,i};
   B = matrix';
   C = reshape(B,1,[]);
   D = runlength_encode(C); % Call runlength_encode function
   rl_array{i,1} = D;
end

%% Apply run length decoding for each residual image

for i = 1:num_frames-1
   D = runlength_decode(rl_array{i,1}); % Call
runlength_decode function
   A = reshape(D,cols,rows)';
   rl_decode_array{1,i} = A;
end
```

```
%% Apply inverse DCT and dequantization for each residual
image

for i = 1: num_frames-1
    Q1 = dequantized_idct(double(rl_decode_array{1,i}),quant);
% Call dequantized_idct function
    inv_DFT{1,i} = Q1;
end

%% Reconstructed frames

for i = 1: num_frames-1
    reconstructed_img{1,i+1} = double(inv_DFT{1,i})+
double(predicted_frames{1,i});
end

% Save reconstructed frames as JPG images
for i = 1:num_frames
 frame_file = strcat('reconstructed_frame',num2str(i),'.jpg');
 frame = uint8(reconstructed_img{1,i});
 imwrite(frame,frame_file);
end


%% Save the reconstructed video

reconstructed_vid = VideoWriter('reconstructed video.mp4');
reconstructed_vid.FrameRate = video.FrameRate;
open(reconstructed_vid);
for j = 1:num_frames
 fram_num = uint8(reconstructed_img{1,j});
 writeVideo(reconstructed_vid,fram_num);
end
% Close the compressed video file
close(reconstructed_vid);


%% Save the output as text file
text1 = fopen('reconstructed video.txt','w');
fprintf(text1, num2str(rl_array_I{1,1}));
fprintf(text1,',');


for i = 1 : num_frames-1
 fprintf(text1, num2str(rl_array{i,1}));
 fprintf(text1,',');
end
fclose(text1);
```

```
function frame1 = padding(frame,N)
    if N > 0
      frame(end+2*N,end+2*N) = 0 ;
      frame1 = frame([end-N+1:end 1:end-N], [end-N+1:end
1:end-N]) ;
    end
end


function macro = divide_into_macroblocks(frame)
        [r,c] = size(frame);

        s = 1;
        for i =1:r/8
            e = 1;
            for j=1:c/8
                  macro{i,j} = frame(s:s+7,e:e+7);
                  e = e + 8;
            end
              s = s + 8;
        end
end



function img = combine_macroblocks(macro, row_img, col_img)
    img = cell2mat(macro);
    img = img(1:row_img, 1:col_img);
end




function Q = quantized_dct(gray_image,jpeg)

    % The size of the grayscale image
    [h, w] = size(gray_image);
    r = h/8;
    c = w/8;
    s = 1;

    for i=1:r
        e = 1;
        for j=1:c

            % Divide grayscale image into 8x8 macroblocks
            macroblock = gray_image(s:s+7,e:e+7);
```

```
            % Convert the pixel values in the current 
macroblocks to double and centre them by subtracting 
128(Halfway between the minimum and maximum possible pixel 
values in an 8-bit grayscale image)
            centre = double(macroblock) - 128;

            % DCT(Discrete Cosine Transform)
            dc = dct2(centre);

            % Quantization
            Q(s:s+7, e:e+7) = round(dc ./ jpeg);

            e = e + 8;
        end
        s = s + 8;
    end

end



function Out = dequantized_idct(constructed_image,jpeg)

    % The size of the input constructed image
    [h, w] = size(constructed_image);
    r = h/8;
    c = w/8;
    s = 1;

    for i=1:r
        e = 1;
        for j=1:c
            qc = constructed_image(s:s+7, e:e+7);

            % Dequantization
            DQ = jpeg .* qc;

            % Inverse DCT(Inverse Discrete Cosine Transform)
            idc = idct2(DQ);

            % Output matrix
            Out(s:s+7, e:e+7) = idc + 128;

            e = e + 8;
        end
        s = s + 8;
    end

end
```

```matlab
function [predict, motion_vec] = motion_vectors(macro)
    num_frames = numel(macro);
    predict = cell(1, num_frames-1);
    motion_vec = cell(1, num_frames-1);

    for g = 2:num_frames
        frame_now = macro{1,g};
        frame_prev = macro{1,g-1};
        r = size(frame_now, 1) * 8;
        c = size(frame_now, 2) * 8;
        pred = cell(r/8-2, c/8-2);
        mv = cell(r/8-2, c/8-2);

        for p = 2:(r/8-1)
            for q = 2:(c/8-1)
                macro_now = frame_now{p,q};
                best_error = Inf;
                best_pred = [];
                best_mv = [];

                for a = (p-1:p+1)
                    for b = (q-1:q+1)
                        macro_prev = frame_prev{a,b};
                        SAD = sum(abs(macro_now(:) -
macro_prev(:)));

                        if SAD < best_error
                            best_error = SAD;
                            best_pred = macro_prev;
                            best_mv = [a,b];
                        end
                    end
                end

                pred{p-1,q-1} = best_pred;
                mv{p-1,q-1} = best_mv;
            end
        end

        predict{g-1} = pred;
        motion_vec{g-1} = mv;
    end
end
```

```matlab
function encoded_data = runlength_encode(quant_matrix)
    % Initialize variables
    encoded_data = [];
    count = 1;
    for i = 1:length(quant_matrix)-1
        % If an item is repeated, increase the count
        if(quant_matrix(i)== quant_matrix(i+1))
            count = count+1;
        else
            encoded_data =
[encoded_data,count,quant_matrix(i),];
        count = 1;  % Reset count
        end
    end
    % Encode the last count and the item
    encoded_data =
[encoded_data,count,quant_matrix(length(quant_matrix))];
end
```

```matlab
function decoded_data = runlength_decode(encoded_data)
    % Initialize variables
    count_values = [];
    symbol_values = [];
    decoded_data = [];

    for i = 1:2:length(encoded_data)
        count_values = [count_values encoded_data(i)];
    end

    for i = 2:2:length(encoded_data)
        symbol_values = [symbol_values encoded_data(i)];
    end

    for i = 1:length(count_values)
        count = count_values(i);
        symbol = symbol_values(i);
        for j = 1:count
            decoded_data = [decoded_data symbol];
        end
    end
end
```

Improved Video Codec

```
%%% Improved Video Codec -> Rate Distortion Optimization %%%

clear all;
clc;

%% Distortion vs Transmission Rate Curve

% transmission_rate =
[32,30,40,36,32,30,28,30,29,28,23,18,18,15,14,12,13,12,11,8,6,
6,5];
% distortion =
[0.022809126,0.024503318,0.025350342,0.025944712,0.026409822,0
.026791551,0.027131078,0.027426941,0.27690783,0.027926642,0.02
9669893,0.030928441,0.031992015,0.032883055,0.033748439,0.0344
90013,0.035140986,0.035990383,0.036590497,0.041847658,0.047279
983,0.050483378,0.055686729];
%
% % Polynomial degree
% degree = 3;  % Adjust the degree as desired
%
% % Perform polynomial fitting
% coefficients1 = polyfit(transmission_rate, distortion ,
degree);
%
% % Generate x-values for plotting
% xmin = min(transmission_rate);
% xmax = max(transmission_rate);
% num_points = 1000000;
% x_plot = linspace(xmin, xmax, num_points);
%
% % Evaluate the fitted polynomial
% y_plot = polyval(coefficients1, x_plot);
%
% % Plot the data points and the fitted curve
% figure;
% plot(transmission_rate, distortion , '*', x_plot, y_plot)
%
% % Add labels and title
% xlabel('Transmission Rate (kbps)')
% ylabel('Distortion')
% title('Distortion vs. Transmission Rate')
% ylim([0.022, 0.06])
%
% % Add legends
% legend('Data Points', '4th Order')

%% Quantization Matrix Multiplication Factor vs Transmission
Rate Curve
```

```
transmission_rate =
[32,30,40,36,32,30,28,30,29,28,23,18,18,15,14,12,13,12,11,8,6,
6,5];
q_matrix_mf =
[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,3,4,5,6,7,8,9,10,20,3
0,40,50];

% Polynomial degree
degree = 4;  % Adjust the degree as desired

% Perform polynomial fitting
coefficients2 = polyfit(transmission_rate, q_matrix_mf ,
degree);

% Generate x-values for plotting
xmin = min(transmission_rate);
xmax = max(transmission_rate);
num_points = 1000;
x_plot = linspace(xmin, xmax, num_points);

% Evaluate the fitted polynomial
y_plot = polyval(coefficients2, x_plot);

% Plot the data points and the fitted curve
figure;
plot(transmission_rate, q_matrix_mf, '*', x_plot, y_plot)

% Add labels and title
xlabel('Transmission Rate (kbps)')
ylabel('Quantization Matrix Multiplication Factor')
title('Quantization Matrix Multiplication Factor vs.
Transmission Rate')

% Add legends
legend('Data Points', '4th Order')
```