1. Introduction

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 develop simplified version of Video coding system using tools like discrete cosine transformation, quantization, prediction, Intra coding and entropy coding.

During this project, block based image compression which can also be used in compressing I frames of video and motion-compensated frame prediction using block matching by evaluating motion vector was studied.

In image compression, DCT (Discrete Cosine Transform) for image transformation and entropy coding techniques; Huffman coding and Run Length coding was implemented. Furthermore, quality of image with different quantization levels also studied.

Then, in video coding, inter coding of P frames using block matching was implemented. Matlab 2015a version was used for coding and implementation purpose.

2. Basic concepts in image compression System

Simplified version of image compression flow chart is given below.

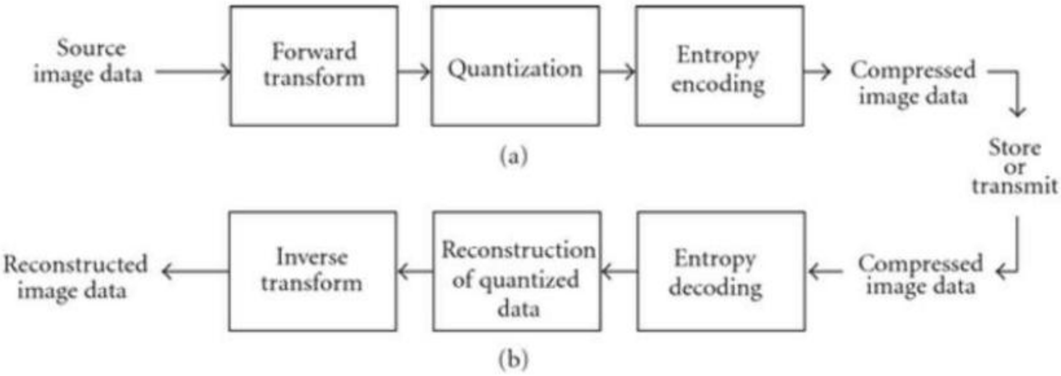


Fig 01: Block diagram of simplified version of image compression

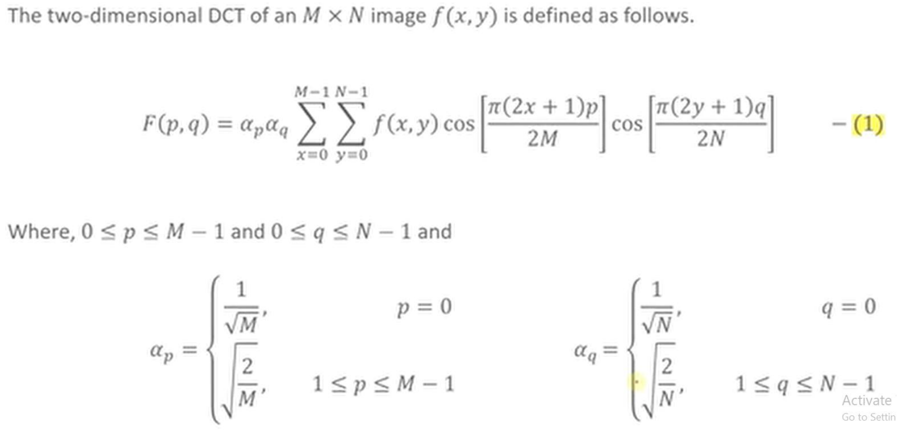
Theory behind techniques corresponding to each part of image compression which is used during the project is discussed below.

2.1 Discrete Cosine Transform (DCT)

The Discrete Cosine transform (DCT) represents an image as a sum of sinusoids of varying magnitudes and frequencies. DCT is derived from extracting real part of corresponding Discrete fourier transform. Since coefficients of DCT are real instead of complex, hardware implementation of DCT is much easier compared to DFT.

The DCT has the property that, for typical image, most of the visually significant imformation about the image is concentrated in just a few coefficients of the DCT. This property is known as Energy Compaction property.

2D Forward DCT is given below.



The two-dimensional DCT of an $M \times N$ image $f(x, y)$ is defined as follows.

$$F(p,q) = \alpha_p \alpha_q \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{\pi(2x+1)p}{2M}\right] \cos\left[\frac{\pi(2y+1)q}{2N}\right] \qquad -(1)$$

Where, $0 \le p \le M-1$ and $0 \le q \le N-1$ and

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}}, & p = 0 \\ \sqrt{\frac{2}{M}}, & 1 \le p \le M-1 \end{cases} \qquad \alpha_q = \begin{cases} \frac{1}{\sqrt{N}}, & q = 0 \\ \sqrt{\frac{2}{N}}, & 1 \le q \le N-1 \end{cases}$$

2.2 Quantization

Quantization is the most elementary form of lossy data compression.When a number x is quantized to L levels, we mean that its value is replaced by (or quantized to) the nearest member of a set of L quantization levels. Here, we consider uniform quantization.

For the uniform quantization used here:

- We define a quantizer range defined by values xmin and xmax

- We divide this range into L equally sized segments, each with size Δ = (xmax−xmin)/ L .

- We place the quantization level for a given segment in the middle of that segment.

For 8 levels of uniform quantized levels above explanation can be depicted as follows.

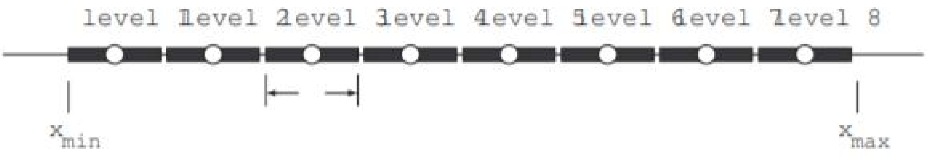
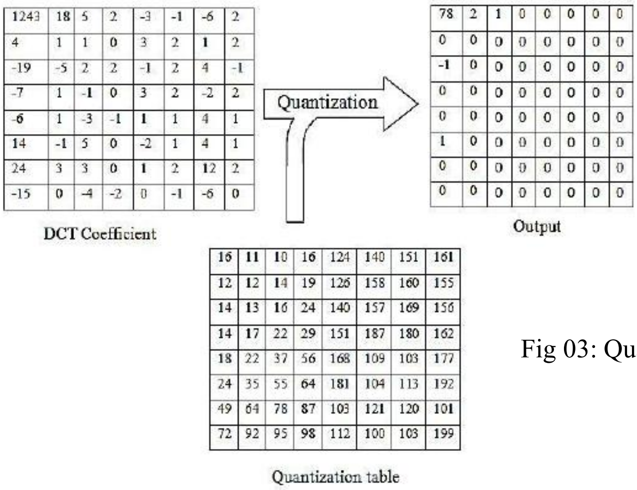


Fig 02: Quantizaton using 8 levels

A typical image compression works by breaking the picture into discrete blocks (8×8 pixels blocks also known as marco blocks). These blocks can then be subjected to discrete cosine transform (DCT) to calculate the frequency components, both horizontally and vertically. The resulting block (the same size as the original block) is then pre-multiplied by the quantization scale code and divided element-wise by the quantization matrix, and rounding each resultant element.

A common quantization process behaves like following


Quantization

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

DCT Coefficient

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

Output

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

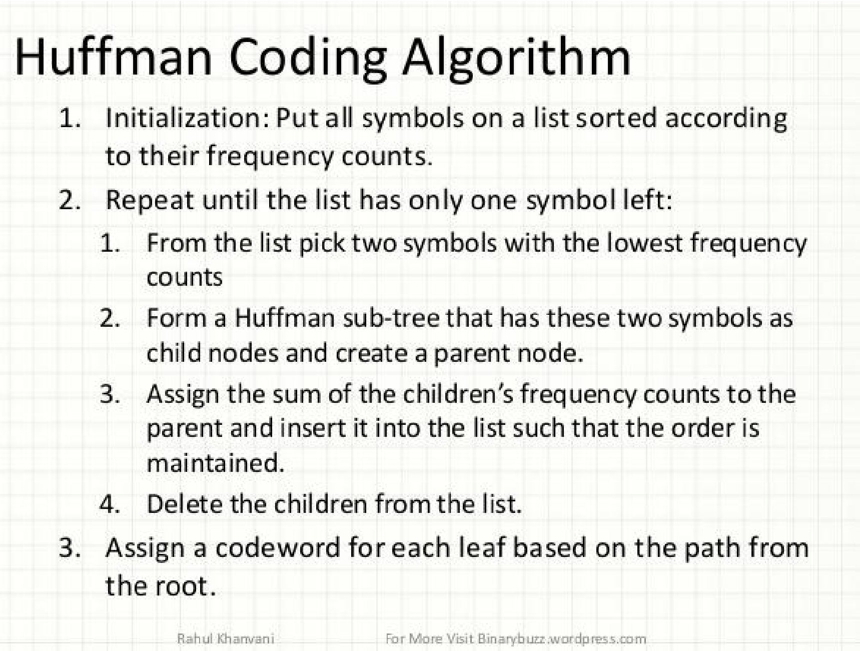
Quantization table

Fig 03: Quantiztion of DCT transform image matrix

2.3 Huffman coding

Huffman coding is a lossless coding technique which was proposed by Dr. David A. Huffman in 1952.It is method for the construction of minimum redundancy code. Huffman code is a technique for compressing data. Huffman's greedy algorithm looks at the occurrence of each character and it as a binary string in an optimal way.

Huffman algorithm accomplish it’s data compression by allowing shorter code words for frequently occuring characters/data. Therefore, code word lengths are no longer fixed like in ASCII.



Huffman Coding Algorithm

1. Initialization: Put all symbols on a list sorted according to their frequency counts.
2. Repeat until the list has only one symbol left:
    1. From the list pick two symbols with the lowest frequency counts
    2. Form a Huffman sub-tree that has these two symbols as child nodes and create a parent node.
    3. Assign the sum of the children’s frequency counts to the parent and insert it into the list such that the order is maintained.
    4. Delete the children from the list.
3. Assign a codeword for each leaf based on the path from the root.

Rahul Khanvani For More Visit Binarybuzz.wordpress.com

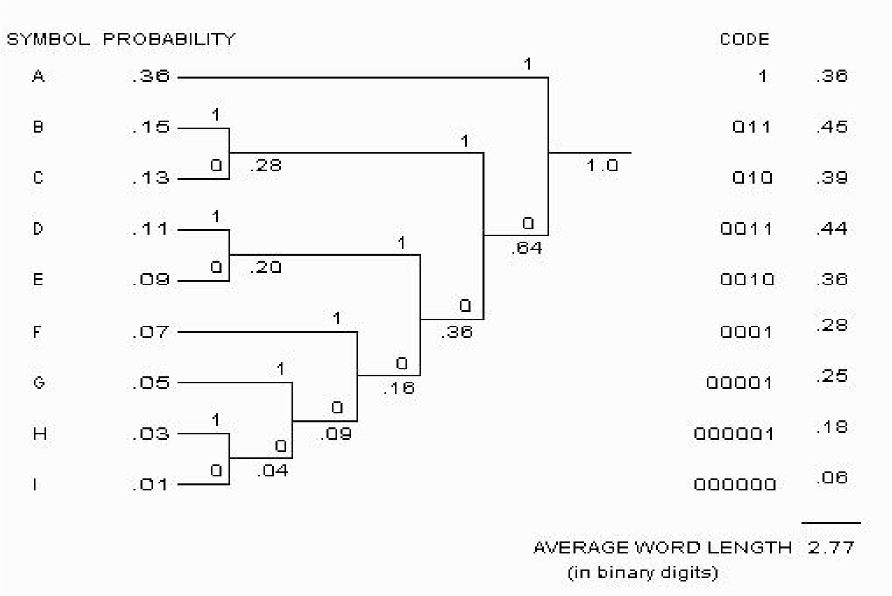


Fig 04: Huffman Coding

2.4 Motion Estimation and Motion Vector

Adjacent frames in a video are quite similar and differ due to the motion of object or camera. Hence, this property of similarity is called as “Temporal correlation”. Due to the above characteristics, when coding adjacent frames, we can use these temporal redundancies to compress frames. In real video scenes, motion can be a complex combination of translation and rotation. Such motion is difficult to estimate and may require large amounts of processing. However, translational motion is easily estimated and has been used successfully for motion compensated coding.

Block matching algorithm is used to obtain motion vector.

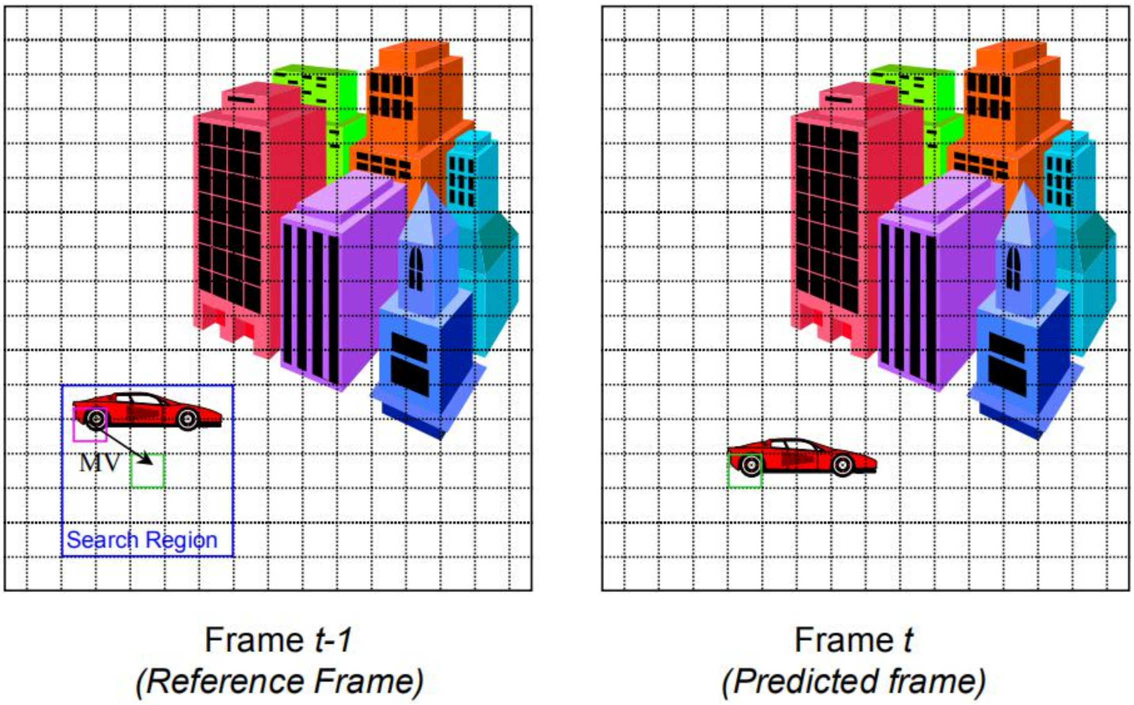


Fig 05 Motion Vector and motion estimation

2.5 Block matching Algorithm

Objective of block matching is to find the best matching marco block for a given frame (Target frame) using reference frame. This algorithm able to calculate Motion Vector generation by calculating x axis and y axis displacement between current marco block and best matched marco block.Current marco block of target frame is replaced with best matched marco block. By continue this for every marco block, motion - compensated image is obtain.Difference between motion-compensated image and current frame is calculated as residue.Using reference frame, motion vector and residue, target frame can be reconstructed.

Criteria to find best matched marco block.

- Mean Square error
- Mean Absolute error
- Sum of absolute error

During this project, Sum of absolute error criteria is used to find best matching marco block.

SAD=sum(sum(abs(f1(i:i+N-1,j:j+N-1)-f2(i+k:i+k+N-1,j+l:j+l+N-1))))

Techniques for searching best matched marco block.

- Full / Sequential search
- Logarithmic search
- Hierachial search

During this project, sequential search technique is used.

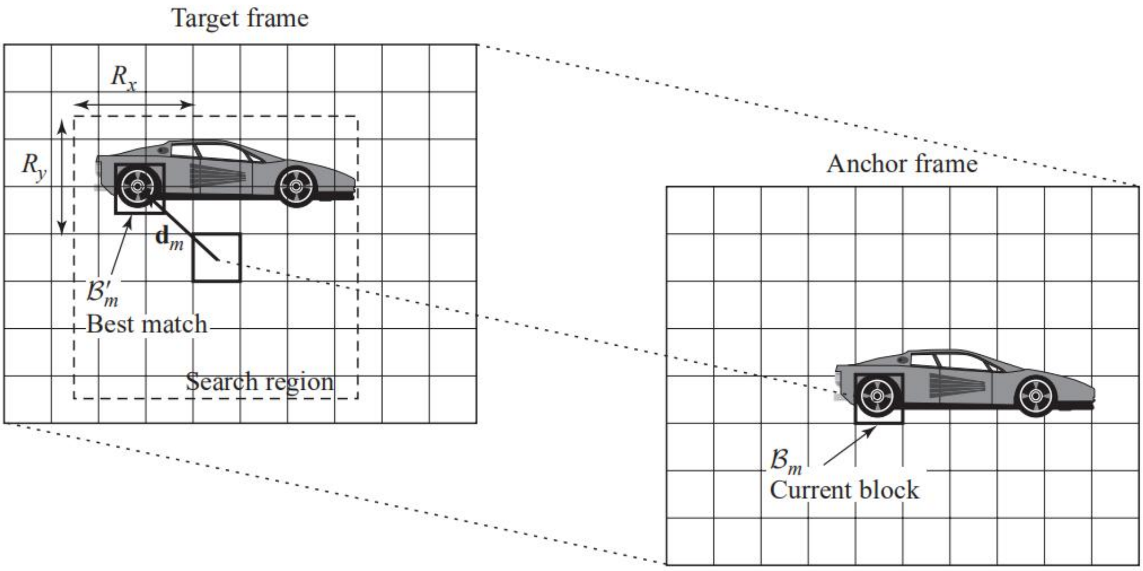
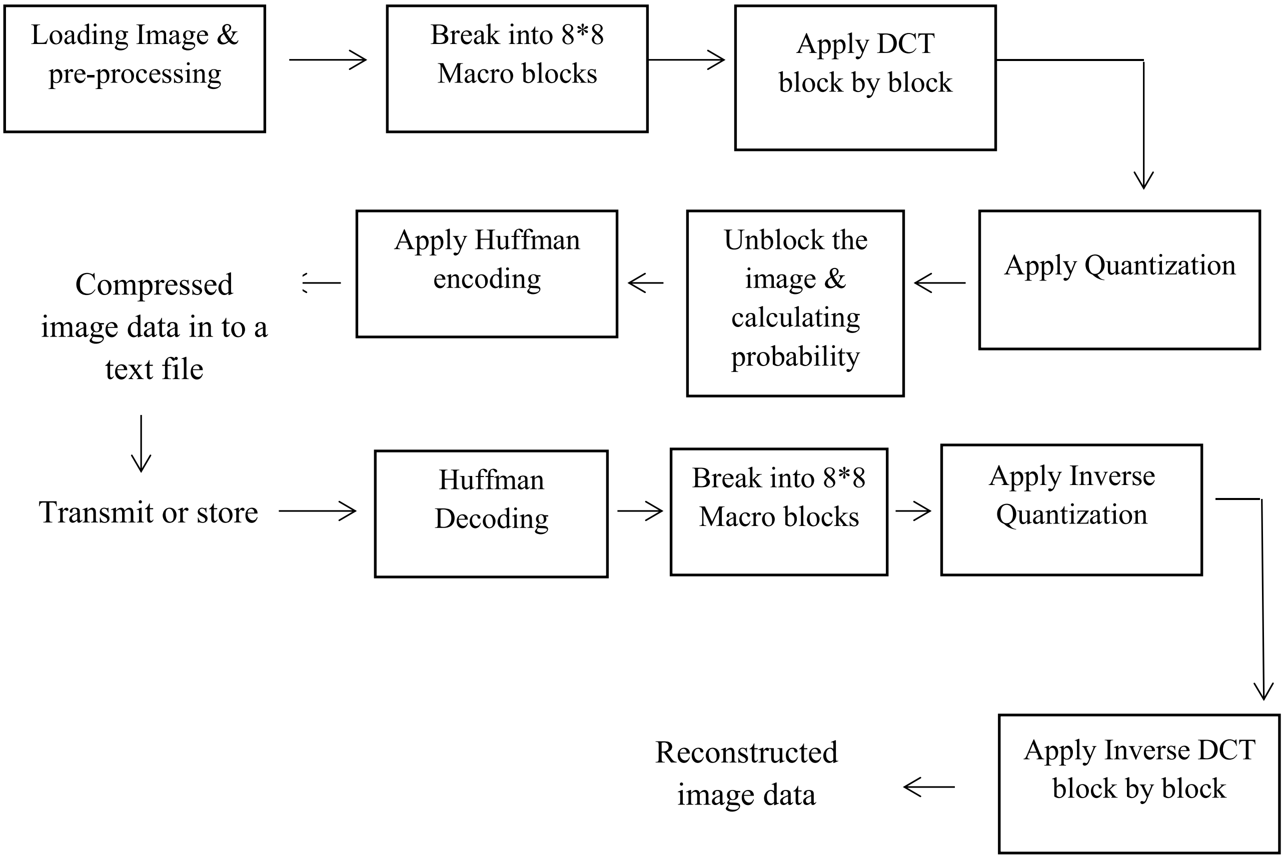


Fig 06: Block matching

3. Image Compression System

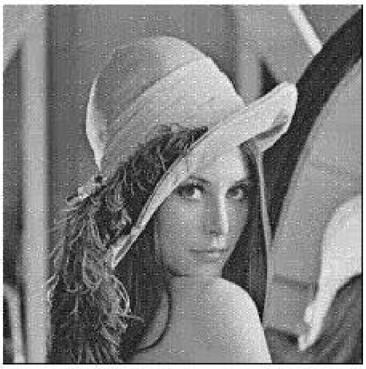
3.1 Flow Chart


Loading Image & pre-processing
Break into 8*8 Macro blocks
Apply DCT block by block
Apply Quantization
Unblock the image & calculating probability
Apply Huffman encoding
Compressed image data in to a text file
Transmit or store
Huffman Decoding
Break into 8*8 Macro blocks
Apply Inverse Quantization
Apply Inverse DCT block by block
Reconstructed image data

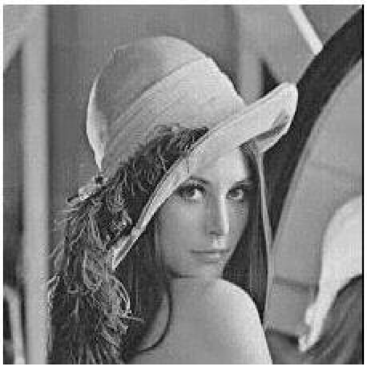
3.2 Results

Original Image

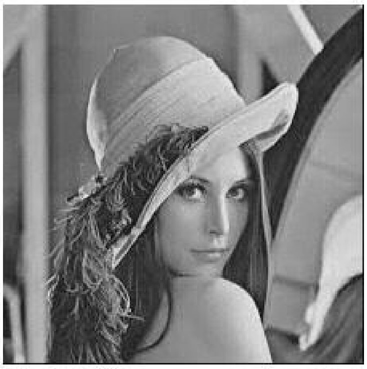
Gray scaled Image



Compressed Image Q_level 1



Compressed Image Q_level 2

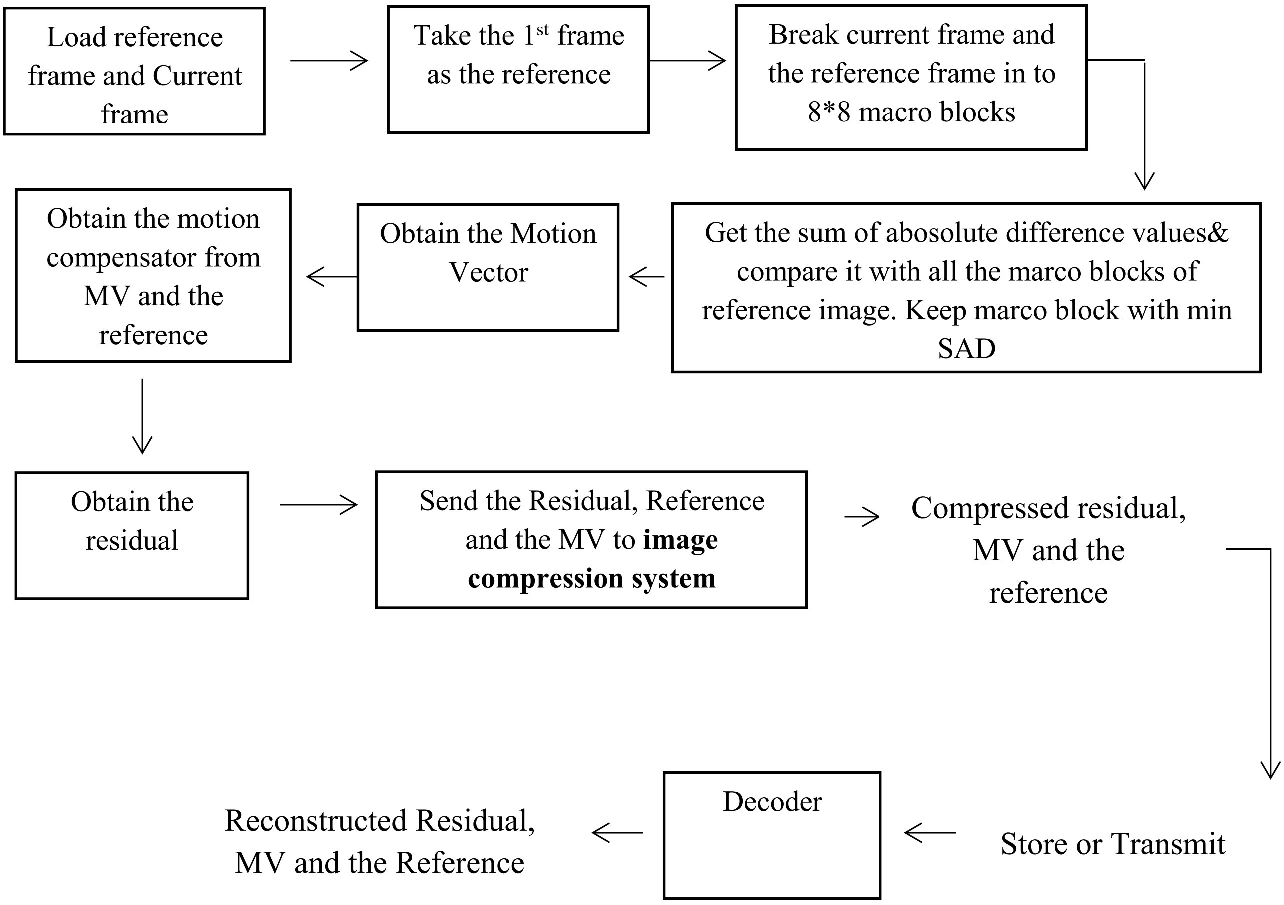


Compressed Image Q_level 3

Video Compression System

This video compression system consists of concepts like Motion Vectors, Motion estimation, Intra prediction frame coding, macro block based coding and etc

Flow Chart



Load reference frame and Current frame

Take the 1st frame as the reference

Break current frame and the reference frame in to 8*8 macro blocks

Get the sum of abosolute difference values& compare it with all the marco blocks of reference image. Keep marco block with min SAD

Obtain the Motion Vector

Obtain the motion compensator from MV and the reference

Obtain the residual

Send the Residual, Reference and the MV to **image compression system**

Compressed residual, MV and the reference

Store or Transmit

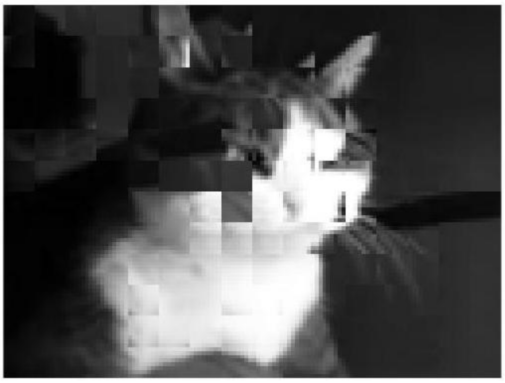
Decoder

Reconstructed Residual, MV and the Reference

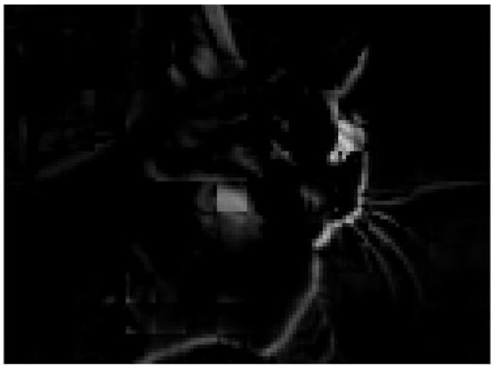
Results

Reference Frame (I Frame)

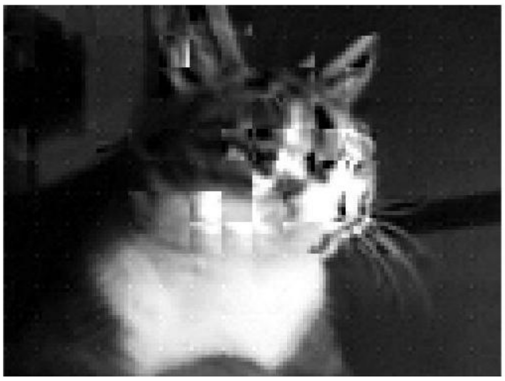
Target Frame (P Frame)



Motion - Compensated Image



Residue



Reconstructed Image

Appendix

1 .Image Compression

```
I=imread('download.jpg');
Gray_I= rgb2gray(I); %converted into gray scaled image
Gray_I = im2double(Gray_I);
%imshow(itg)
imhist(Gray_I);hold on;
%%
%representing discrete cosine transform using 8 by 8 blocks
B = blkproc(Gray_I,[8 8],'dct2');
B = ceil(B*1000);
%%
%%
%quantization values for each quality level
q_low = 8;
q_medium = 16;
q_high = 8;
%performing quantization
Blow = B/(2*q_low);
Blow = ceil(Blow);
%%
%calculate frequency and its probability for each quantized value
[glow,~,intensity_val] = grp2idx(Blow(:));
Frequency = accumarray(glow,1);
probability = Frequency./(225*225);
T = table(intensity_val ,Frequency,probability);
%%
%perform huffman coding
dict=huffmandict(intensity_val,probability);
encode_I = huffmanenco(Blow(:),dict);
```

```
%%

%perform huffman decoding
decode_I = huffmandeco(encode_I,dict);
re_img = reshape(decode_I,[225 225]); %reshaping the image
%%
%dequantization
Blow2 = re_img*q_low*2;
Blow2 = Blow2/1000;
I_decoded = blkproc(Blow2,[8 8],'idct2'); %performing inverse dct
imshow(Gray_I),figure,imshow(I_decoded);
%%
imhist(I_decoded);
%%
%perform run length coding
[d, c ] = my_RLE(Blow(:));
%%
decode_I2 = rl_dec(d,c);
re_img2 = reshape(decode_I2,[225 225]);
imshow(re_img2);
%%
%dequantization
Blow3 = re_img2*q_medium*2;
Blow3 = Blow3/1000;
I_decoded2 = blkproc(Blow2,[8 8],'idct2'); %performing inverse dct
imshow(Gray_I),figure,imshow(I_decoded2);
```

2. Run length Encoding

```
function [d,c]=my_RLE(x);

% This function performs Run Length Encoding to a strem of data x.

% [d,c]=rl_enc(x) returns the element values in d and their number of

% apperance in c. All number formats are accepted for the elements of x.


if nargin~=1

   error('A single 1-D stream must be used as an input')

end

ind=1;

d(ind)=x(1);

c(ind)=1;

for i=2 :length(x)

   if x(i-1)==x(i)

      c(ind)=c(ind)+1;

   else ind=ind+1;

        d(ind)=x(i);

        c(ind)=1;

   end

end
```

3. Run Length Decoding

```
function x=rl_dec(d,c);
% This function performs Run Length Dencoding to the elements of the strem
% of data d according to their number of apperance given in c. There is no
% restriction on the format of the elements of d, while the elements of c
% must all be integers.



if nargin<2
   error('not enough number of inputs')
end
x=[];
for i=1:length(d)
x=[x d(i)*ones(1,c(i))];
end
```

4. Motion Vecotor and residue Calculaton

```
%%
%f1: anchor frame; f2: target frame, fp: predicted image;
%mvx,mvy: store the MV image
%widthxheight: image size; N: block size, R: search range

function [vector,residue] = motionVector(f1,f2,N)
S = size(f1);
height = S(1);
width = S(2);
%N = 8;
%R = -50:50;
vectors = zeros(2,floor(height/N)*floor(width/N));
blk_count = 1;
%%
```

```
for i=1:N:height-N+1

for j=1:N:width-N+1 %for every block in the anchor frame


MAD_min=256*N*N;

mvx=0;

mvy=0;


for k= 1:1:height-N+1

for l= 1:1:width-N+1 %for every search candidate

refBlkVer = i + k; % row/Vert co-ordinate for ref block

refBlkHor = j + l; % col/Horizontal co-ordinate

if ( refBlkVer < 1 || refBlkVer+N-1 > height || refBlkHor < 1 || refBlkHor+N-1 >
width)

continue;

end

MAD=sum(sum(abs(f1(i:i+N-1,j:j+N-1)-f2(i+k:i+k+N-1,j+l:j+l+N-1))));

% calculate MAD for this candidate

if MAD<MAD_min

MAD_min=MAD;dy=k;dx=l;

end;

end;

end;

fp(i:i+N-1,j:j+N-1)= f2(i+dy:i+dy+N-1,j+dx:j+dx+N-1);

%put the best matching block in the predicted image

vectors(1,blk_count) = dy; % row co-ordinate for the vector

vectors(2,blk_count) = dx; % col co-ordinate for the vector

blk_count = blk_count+1;

end;
```

```
end;

vector = vectors;

S2 = size(fp);

residue = f2(1:S2(1),1:S2(2)) - fp;
```

5. Motion compensated image contruction

```
% Computes motion compensated image using the given motion vectors

% Input

% I : The reference image

% motionVect : The motion vectors

% block_size : Size of the macroblock

% Ouput

% imgComp : The motion compensated image

%



function imgComp = Motion_Compensated_Image(I, motionVect,N)

[height width] = size(I);

block_Count = 1;

for i = 1:N:height-N+1

 for j = 1:N:width-N+1



 % dy is row(vertical) index

 % dx is col(horizontal) index

 % this means we are scanning in order



 dy = motionVect(1,block_Count);

 dx = motionVect(2,block_Count);

 refBlkVer = i + dy;

 refBlkHor = j + dx;

 imageComp(i:i+N-1,j:j+N-1) =
I(refBlkVer:refBlkVer+N-1,refBlkHor:refBlkHor+N-1);
```

```
 block_Count = block_Count + 1;
 end
end
imgComp = imageComp
```

6. Video Compression

```
%%
f1 = imread('cat1.jpg');
f2 = imread('cat2.jpg');
f1= rgb2gray(f1); %converted into gray scaled image
f1 = im2double(f1);
f1 = imresize(f1, [100 130]);
f2= rgb2gray(f2);
f2 = im2double(f2);
f2 = imresize(f2, [100 130]);
%%
[Motion_vector,residue] = motionVector(f1,f2,8); %Calculating Motion vector
%%

S = size(residue);
%%
B = blkproc(residue,[8 8],'dct2');
B = ceil(B*1000);

%%

%quantization values for each quality level
q_low = 32;
q_medium = 8;
q_high = 8;
```

```
%performing quantization

Blow = B/(2*q_medium);

Blow = ceil(Blow);

%%


%calculate frequency and its probability for each quantized value

[glow,~,intensity_val] = grp2idx(Blow(:));

Frequency = accumarray(glow,1);

probability = Frequency./(S(1)*S(2));

T = table(intensity_val ,Frequency,probability);

%%


%perform huffman coding

dict=huffmandict(intensity_val,probability);

encode_I = huffmanenco(Blow(:),dict);


%%

[d,c] = my_RLE(Blow(:));


%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Transmision%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%

%%

compensated_img = Motion_Compensated_Image(f1,Motion_vector,8);




%%

decode_I = huffmandeco(encode_I,dict);

re_img = reshape(decode_I,[S(1) S(2)]); %reshaping the image
```

```
%%
re_img = RLE_dec(d,c);
re_img = reshape(re_img,[S(1) S(2)]);

%%

%dequantization
Blow2 = re_img*q_medium*2;
Blow2 = Blow2/1000;

I_decoded = blkproc(Blow2,[8 8],'idct2'); %performing inverse dct

%%
I_decode_dash = I_decoded(1:96,1:128);
%Residue_dash = Residue(1:96,1:128);
%%
compensated_img = im2double(compensated_img);
%Residue_dash = im2double(Residue_dash);
Recontruct_img = compensated_img + I_decode_dash;
%Recontruct_img2 = compensated_img + Residue_dash;
```

Refferences:

1. EE4414: Motion Estimation Basics, Yao Wang, 2003

2. STILL IMAGE AND VIDEO COMPRESSION WITH MATLAB, K. S. Thyagarajan,John willey & sons Publications 2011

3. Aroh Barjatya (2020). Block Matching Algorithms for Motion Estimation (https://www.mathworks.com/matlabcentral/fileexchange/8761-block-matching-algorithms-for-motion-estimation), MATLAB Central File Exchange. Retrieved December 9, 2020.