EE 596 – IMAGE AND VIDEO CODING

MINI PROJECT REPORT

MATARAARACHCHI M.A.R.P.

E/15/224

SEMESTER 07

08.12.2020

Stage 1: Basic Implementation: Image Compression

1. Import Image file from the directory.

   - File name: ‘pic.bmp’
   - Resolution : 1280x720 (HD)
   - Number of colour channels : 1 (Grayscale)

2. Subtract 128 from all the pixels in the image.
   This reduces the DC component of the DCT transformed image by 1024. Therefore, less number of bits are needed to store DC components. This operation can be reversed.

3. Separate the image into 8x8 Marco blocks. Then store the blocks in a cell array.

4. Transform each Marco block using Discrete Cosine Transformation. This step transfers the image from the spatial domain to the frequency domain.

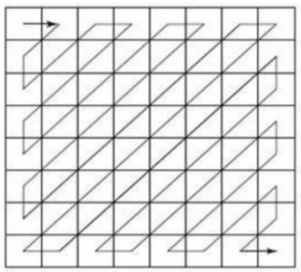
   DCT transform: $C \times M \times C^T$; where C is the 8x8 DCT matrix and M is a Marco block.

5. Quantize each block by dividing it by the standard JPEG quantization matrix. Additionally, divide each block by user defined **quantization level.** As the quantization level increases the quality of the decoded image decreases. This quantization step allows AC values to decrease more than the DC values. This is due to the fact that human eye is more sensitive to DC components of an image than its AC components. This process is analogues to transferring the image through a low pass filter (High frequency components are removed).

```
Quantization 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];
```

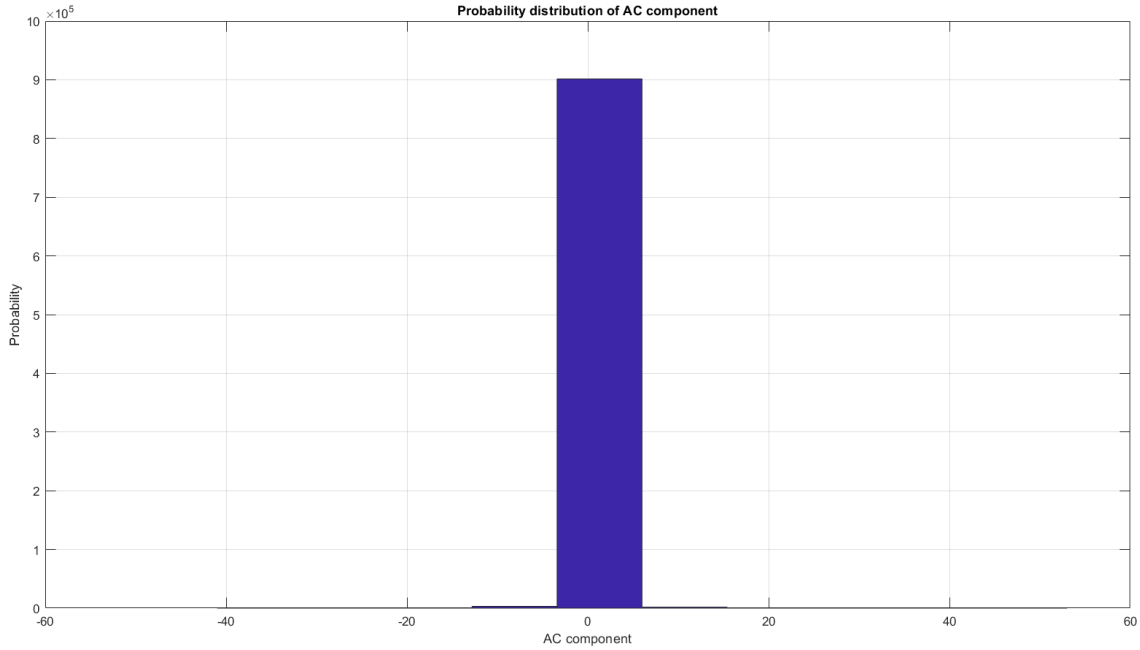
6. Get the ceiling values of all the pixels. This is because we can't encode fractional values in the Huffman code.

7. Zigzag transform the Marco blocks. Since now the most of the AC values in the latter part of the Marco block is zero it is best to get the zigzag transform of all the blocks. This allows more consecutive zeros to be stored in the tail of each block.



8. Store the DC and AC components separately in two arrays.

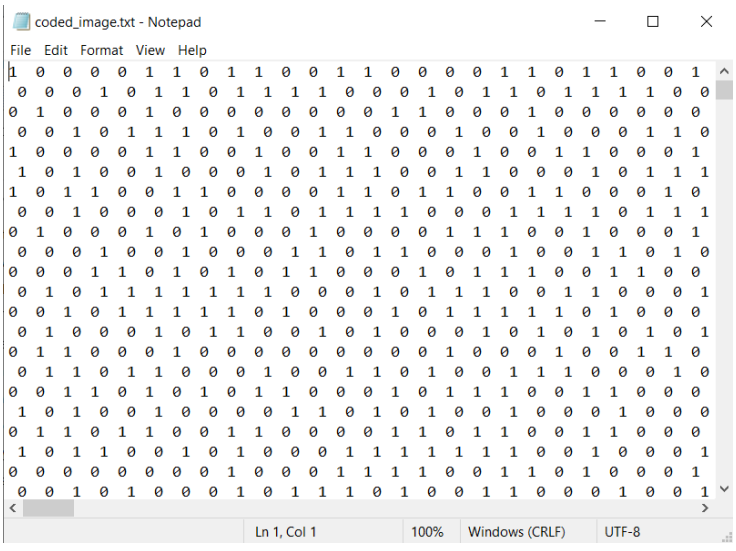
9. As seen from the below histogram, the number of occurrences of ‘0’ is very common in the AC array. Therefore, Run Length encoding is used to encode the AC values separately.



10. Concatenate DC array and Run Length encoded AC array into one array. The first part of this array consists of DC values, and the latter are AC values.

11. Huffman encode the above array using the previously built dictionary generator and Huffman encoder.

12. Store the encoded image into a text file.



Decoding

13. Read the encoded bit sequence from the text file.

14. Decode the encoded bit sequence using the previously built Huffman decoder.

15. Separate the data into DC and AC values.

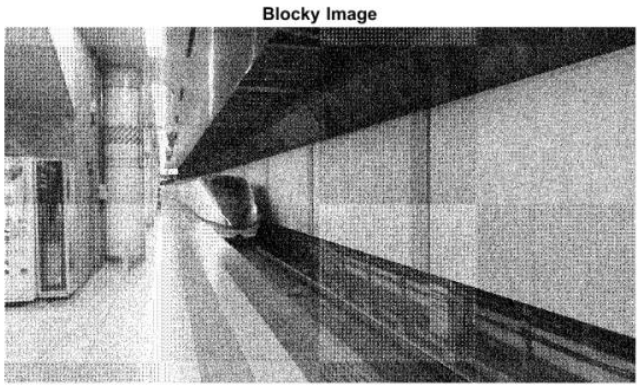
16. Using the DC and AC values build 8x8 Marco blocks. Store them in a cell array.

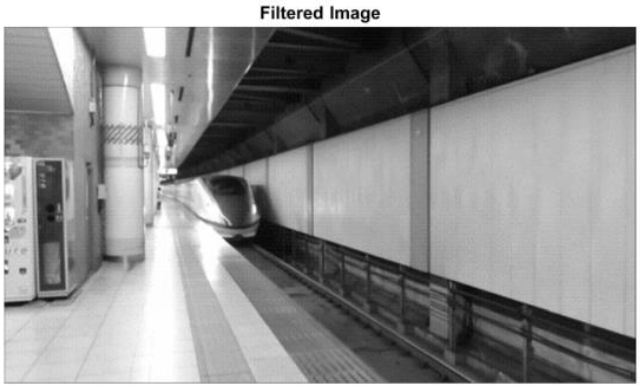
17. Run length decode AC values and Inverse Zigzag transform all the blocks.

18. Inverse quantized all the blocks using the same standard JPEG matrix and the same quantization level.

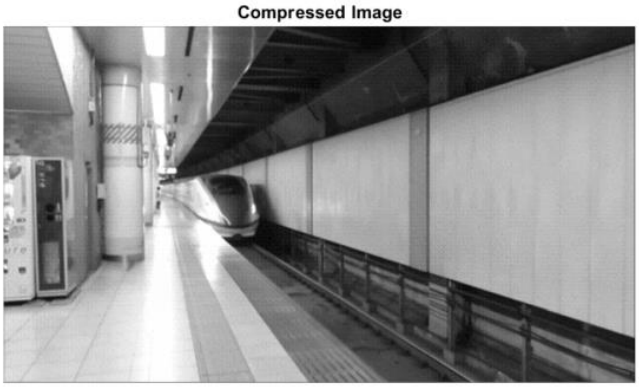
19. Inverse DCT transform all the blocks. Add 128 to all the pixels afterwards.
    Inverse DCT: $C^T \times M \times C$

20. The resulting picture is blocky due to transformations and quantization. Therefore, a filter is needed to get rid of blocky artifacts. Apply averaging filter with an intensity level of 4. This filter type and intensity level was selected after subjectively and objectively (psnr value) evaluating filter outputs.





21. Following image shows a side by side comparison of the uncompressed image and the compressed image.



22. Number of bits before and after compression, Compression ratio.

```
Bits before compression: 7.3728 Mbits
Bits after compression: 2.2043 Mbits
Compression Ratio: 3.3448
Quantization Level: 1
Elapsed time is 6859.578643 seconds.
>>
```

As seen from the metrics, the elapsed time is very large. Therefore, huffmanenco() and huffmandeco() functions are used instead of the custom made encoder and decoder hereafter. **Note: The Huffman dictionary is generated same as before by a custom code.**

Stage 2: Basic Implementation 2: Video Compression

1. Import movie from the directory.
   - File name: 'train.avi'
   - Resolution: 1280x720 (HD)
   - Frame Rate: 30 fps
   - Number of colour channel: 1 (Grayscale)
   - Length: 1s

2. Divide the frames into GOPs (Group of Pictures). This can be adjusted by the *pframes* variable. *pframes* is the number of P frames that follow one I frame.

   In this demonstration, *pframes* = 4.

   Therefore, total frames in a GOP are, 4+1 = 5. (1 I frame and 4 P frames)

   Number of GOPs = 30/5 = 6.

3. Remove temporal redundancies in each GOP. This is done by motion estimation. Each P frame is predicted by the corresponding I frame in the GOP.

   - Marco block size: 8x8 pixels
   - Search area: 24x24 pixels (9 Marco Blocks)
   - I frame (Reference frame) is zero padded to match the dimensions
   - Predicted using SAD (Sum of Absolute difference)
   - Search method used: Sequential Search

   I frame – Reference Picture, P frame – Target Picture.

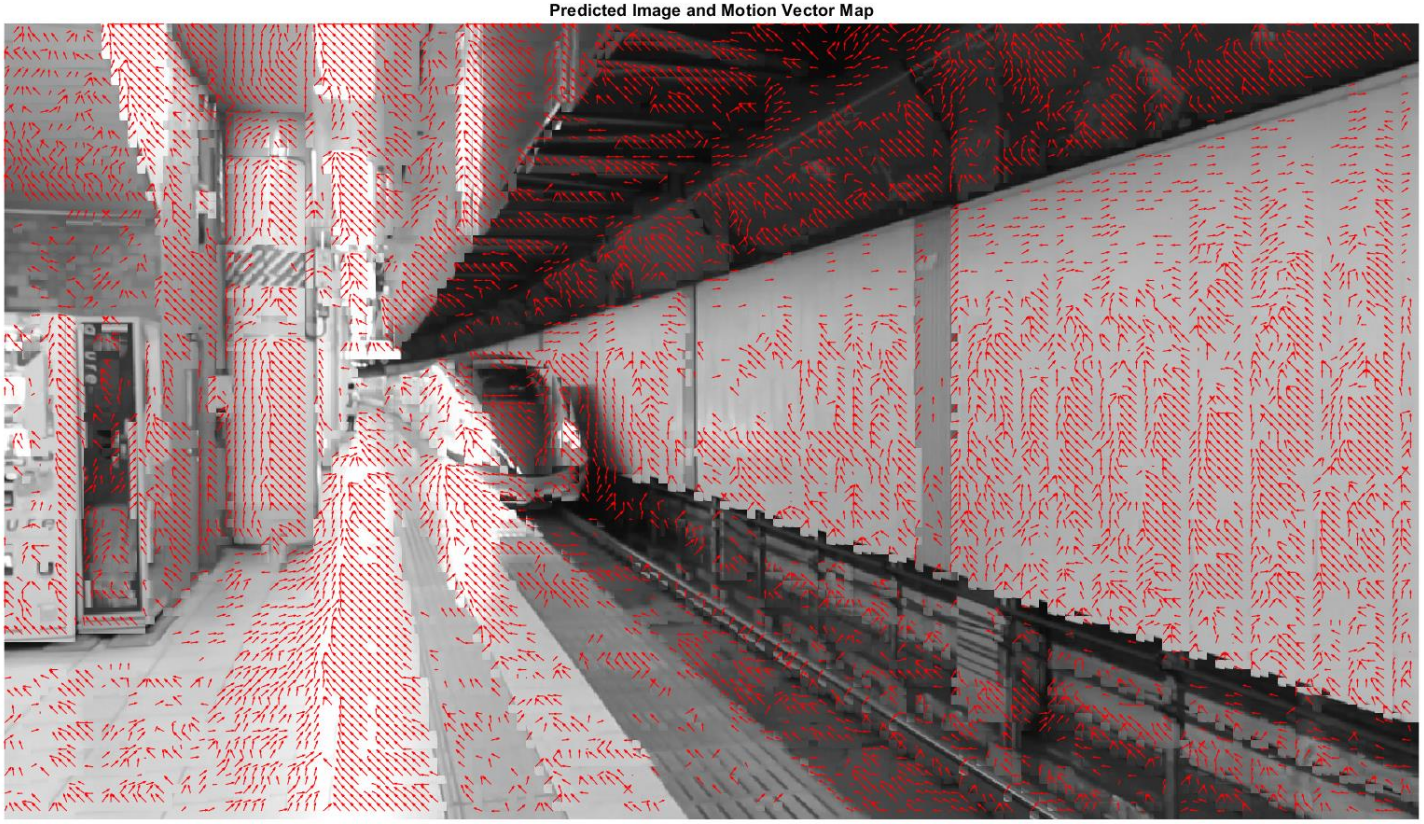
Refernce Picture

Target Picture

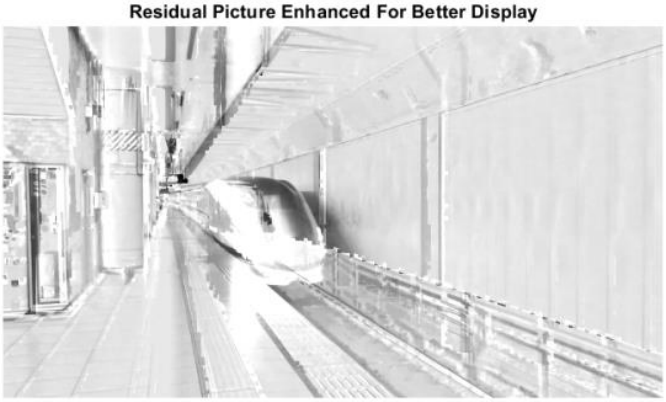
4. An image is predicted using the motion vectors.

Predicted Image

Predicted Image and Motion Vector Map



5. After predicting the target image the difference between the target image and the prediction is taken. This is called the residual.



6. Residual image is coded using Huffman coding after going through DCT, Quantization, Zigzag transform and Run length coding. The motion vector is coded using Huffman coding without quantization (should be lossless).

7. Store all the encoded files as text files.

8. For decoding read the data from text files.

9. Reverse all the steps backwards to decode the video. Huffman decode, Run length decode, inverse zigzag, Inverse quantize, IDCT, add spatial/ temporal redundancies.

10. Write movie to .avi file.

Stage 3: Improved Hybrid Video Codec

Optimize the quantization process of the codec to facilitate transmission in fixed bandwidth environment.

1. Import movie from the directory.
   - File name: 'limes.avi'
   - Resolution: 640x360 (SD)
   - Frame Rate: 10 fps
   - Number of colour channel: 1 (Grayscale)
   - Length: 1s
   - GOP: [IPPPP]
   - Uncompressed bitrate: 18.432 Mbps (= 640*360*8*10/1000000)

2. Bitrate calculation. All the bits that should be transmitted in 1 second should be included in the calculation. This includes the Huffman encoded residuals, Motion vectors, Huffman dictionaries and length arrays (DC length, Motion vector length, etc.)

3. Function *OptimizeQ.m* was written to calculate the bitrate for a given Quantization level (Q).

4. *FixedBRopt.m* was written to optimize the Quantization level for a fixed bitrate. A binary search method was used to find the optimum quantization level for a given bitrate. The algorithm uses an exponentially decreasing step size to find the optimum solution and terminates upon reaching a fixed number of iterations or when the error becomes less than 0.01.

5. Optimized quantization level for the fixed bitrate is given as an output.

Note: Actual bitrate given in the below plot was calculated beforehand for demonstrational purposes. The matlab code does not calculate bitrate values for all Q values; it uses the algorithm to find the best match in the given number of iterations.

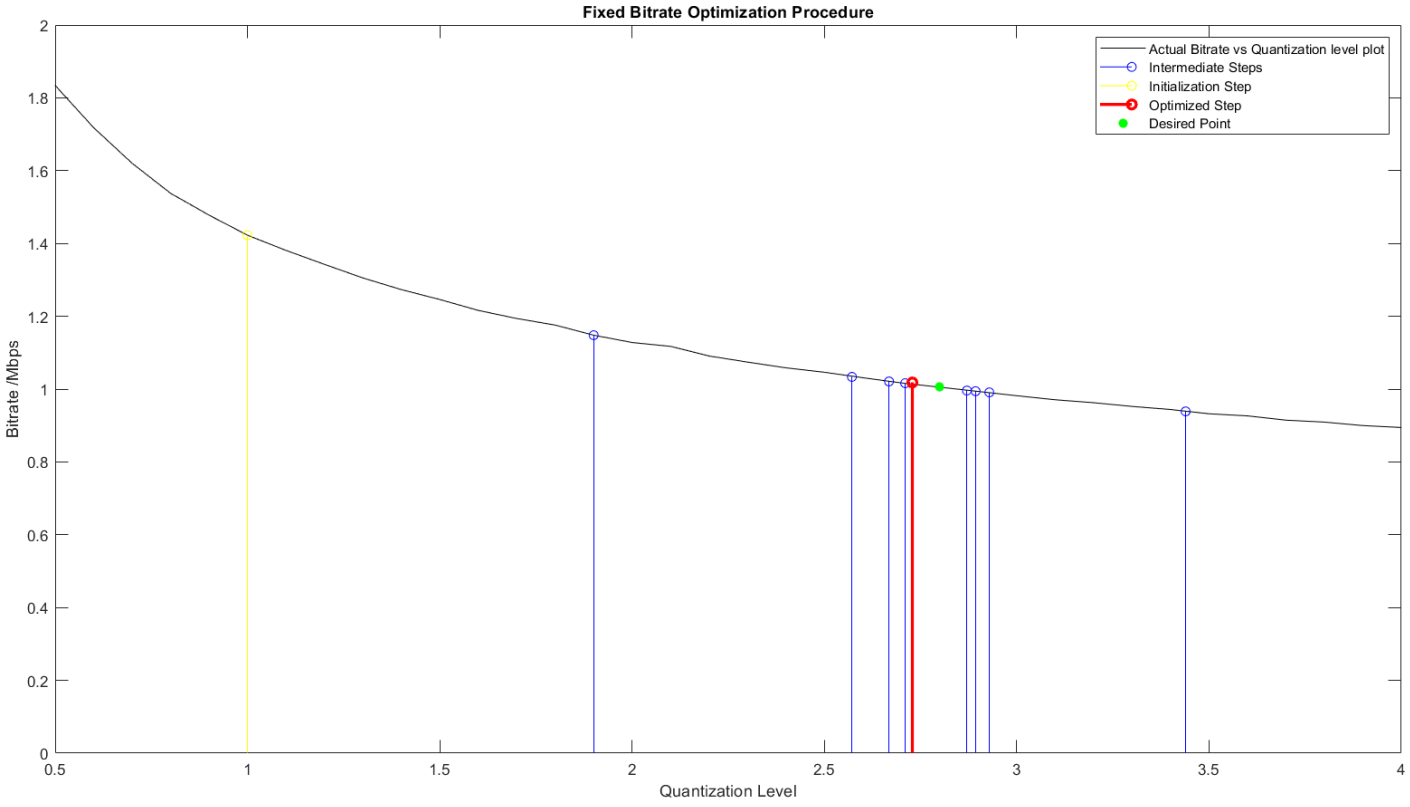
As seen from the below plots the optimization algorithm quickly searches the desired point and the error is very small.

Eg: 01:

Fixed Bitrate: 1 Mbps

Optimized Quantization Level: 2.72870

Bitrate at the optimum quantization level: 1.018 Mbps (Error = +0.018Mbps)

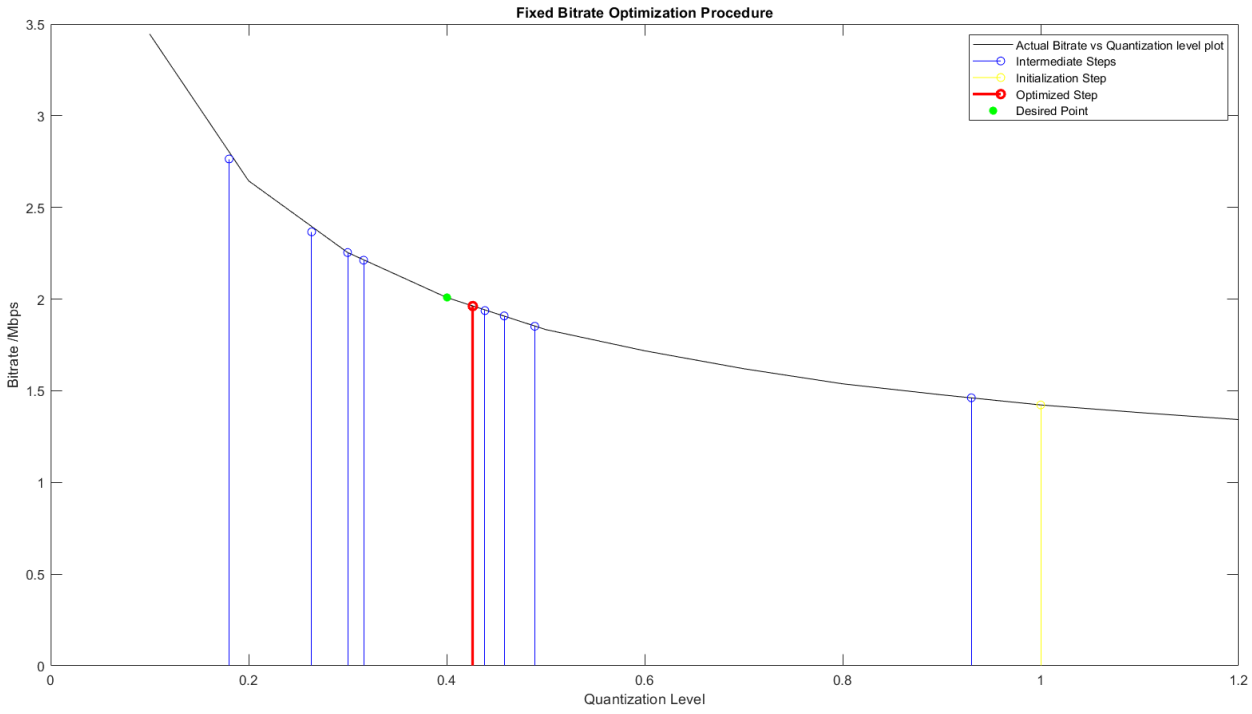


Eg: 02:

Fixed Bitrate: 2 Mbps

Optimized Quantization Level: 0.42643

Bitrate at the optimum quantization level: 1.962 Mbps (Error: -0.038Mbps)



Optimize the coding process to give the best bit-rate for a given quality level.

1. Import Video from Directory (Specifications same as before).

2. A metric to measure the quality is needed. Use VQM – Visual Quality Metric (An Objective Video Quality Measure) to evaluate the quality.

$$VQM = \frac{1}{1+e^{0.17\times PSNR-25.66}} \quad ; 10 \leq PSNR \leq 55. \quad [1]$$

VQM is a measure of distortion and is in the range of [0,1].

VQM is related to quality by,

$$Quality = 100 - 100 \times VQM.$$

VQM was chosen because,

"This metric evaluates the video streams between original video to processed video and includes features with spatial-temporal alignments, valid region-estimations, and gain-level offset calculations. VQM calculation can extract perception-based features to compute the video quality parameters and generate more accurate predictions of human-perceived video quality" [2]

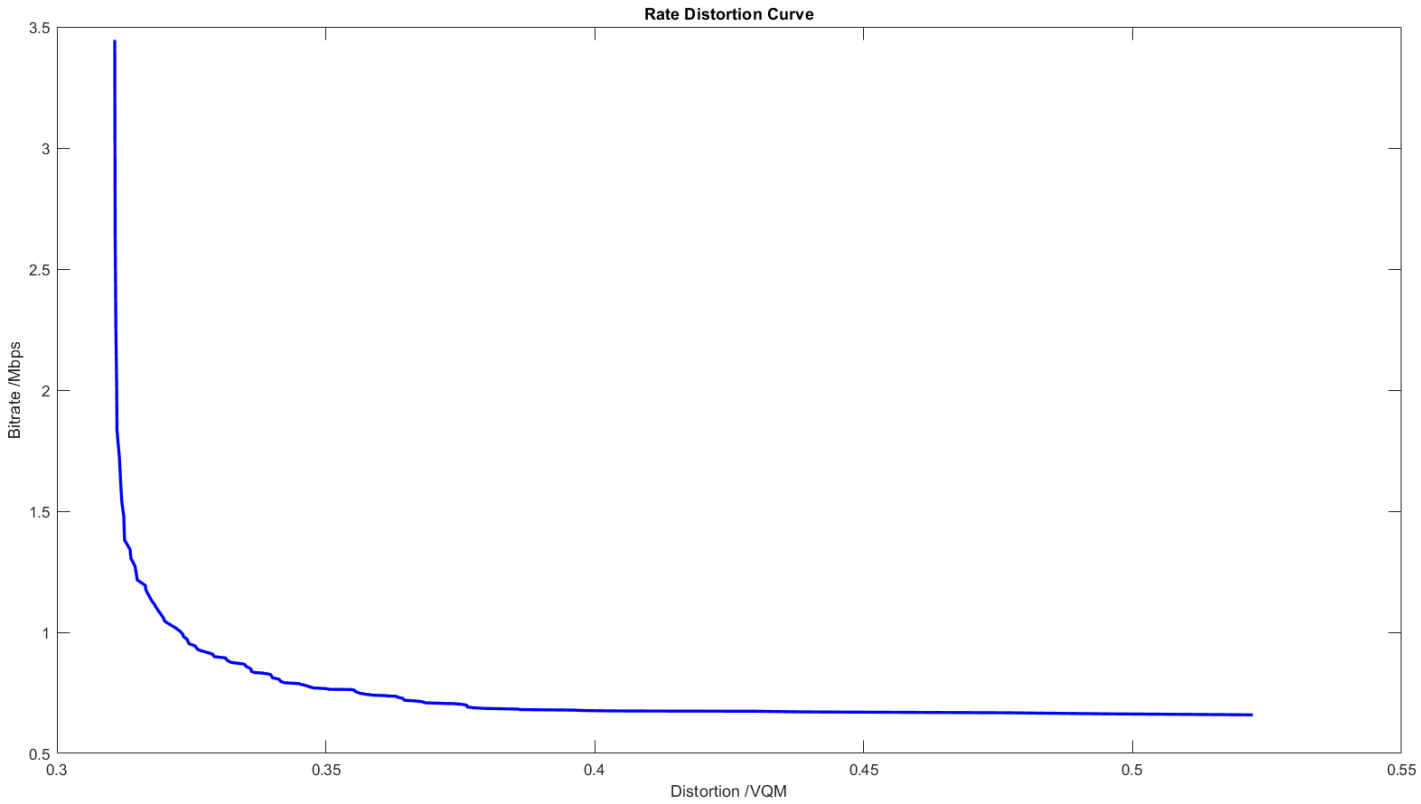
3. Function *OptimizeBR.m* was written to calculate the average VQM over all the frames and bitrate for a given Quantization level (Q). This function encodes the video and decodes it to measure the VQM. This decoding step acts as a feedback loop.

6. *FixedQLTYopt.m* was written to optimize encoding/ decoding to match a given quality level. A binary search method was used to find the optimum bitrate for a given quality. The algorithm uses an exponentially decreasing step size to find the optimum solution and terminates upon reaching a fixed number of iterations or when the error becomes less than 0.001.

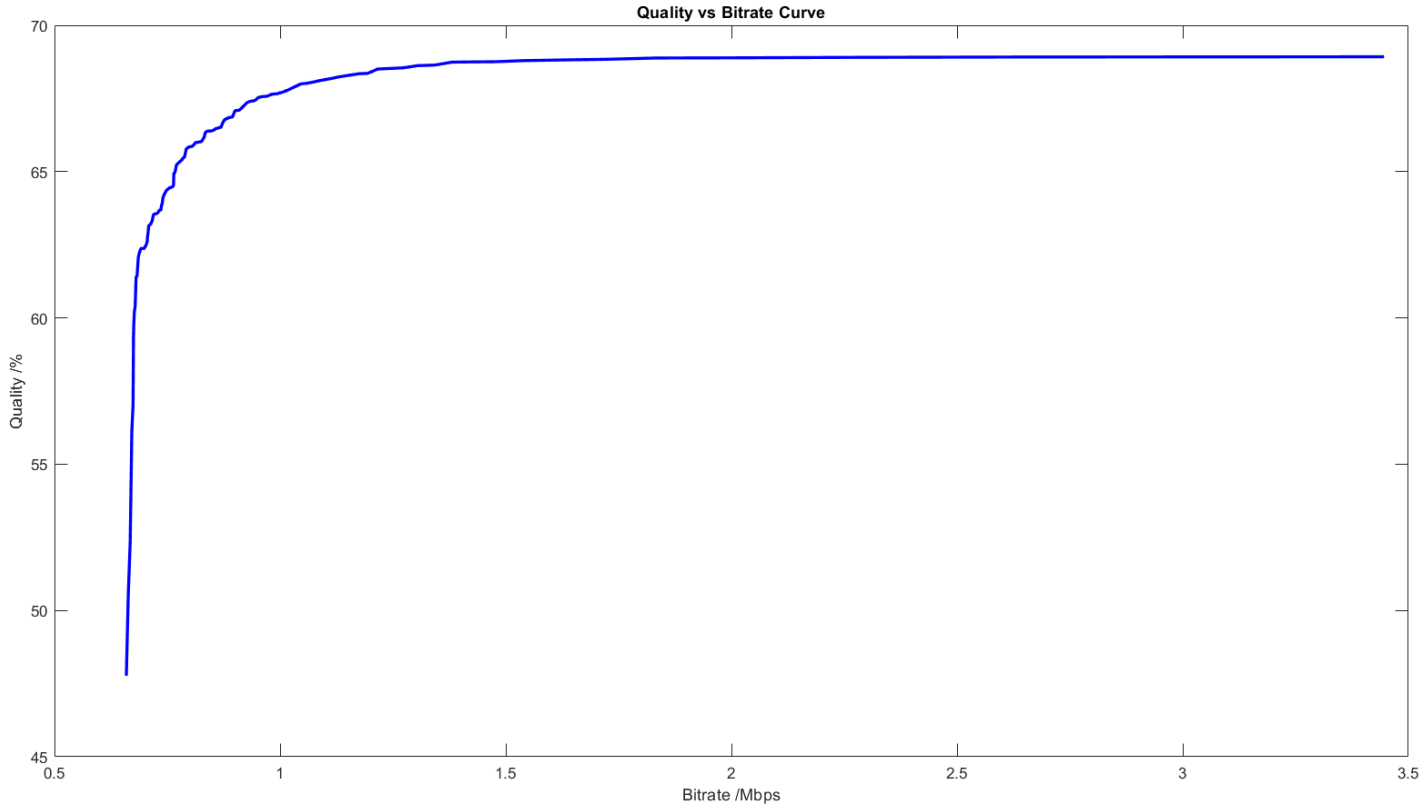
7. Optimized video and bitrate for a given quality level is given as an output.

Note: An approximated Rate Distortion curve was calculated beforehand for demonstrational purposes. The optimization procedure doesn't calculate the bitrate for all the VQM (disortion) values or Quality values. It uses the algorithm to find the best match in the given number of iterations.

Approximated Rate-Distortion Curve.



Approximated Quality vs Bitrate curve



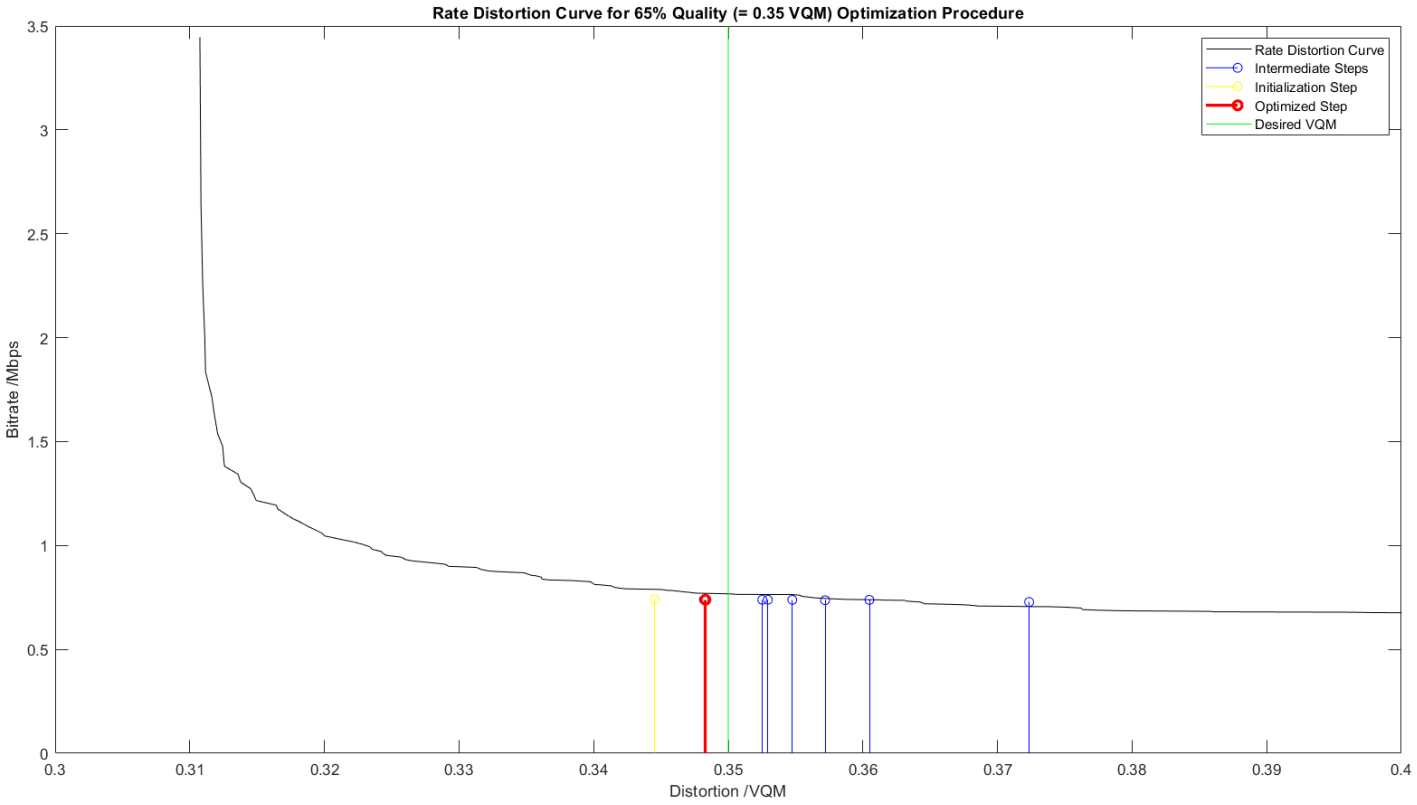
Eg 01:

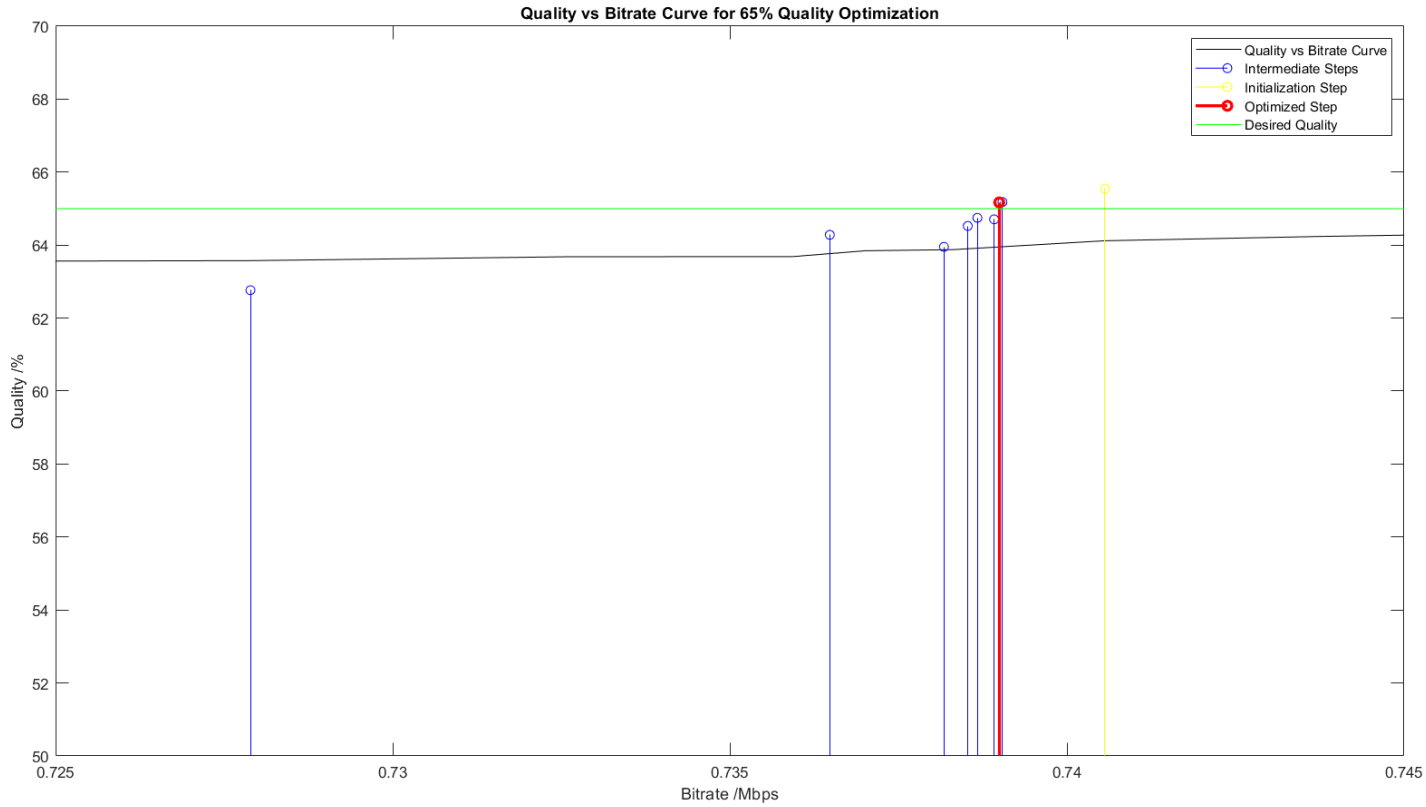
Fixed Quality: 65% (=0.35 VQM)

Optimized bitrate: 0.739 Mbps

Quality at optimized level: 65.170% (Error: +0.17%)

Quantization level: 7.06836





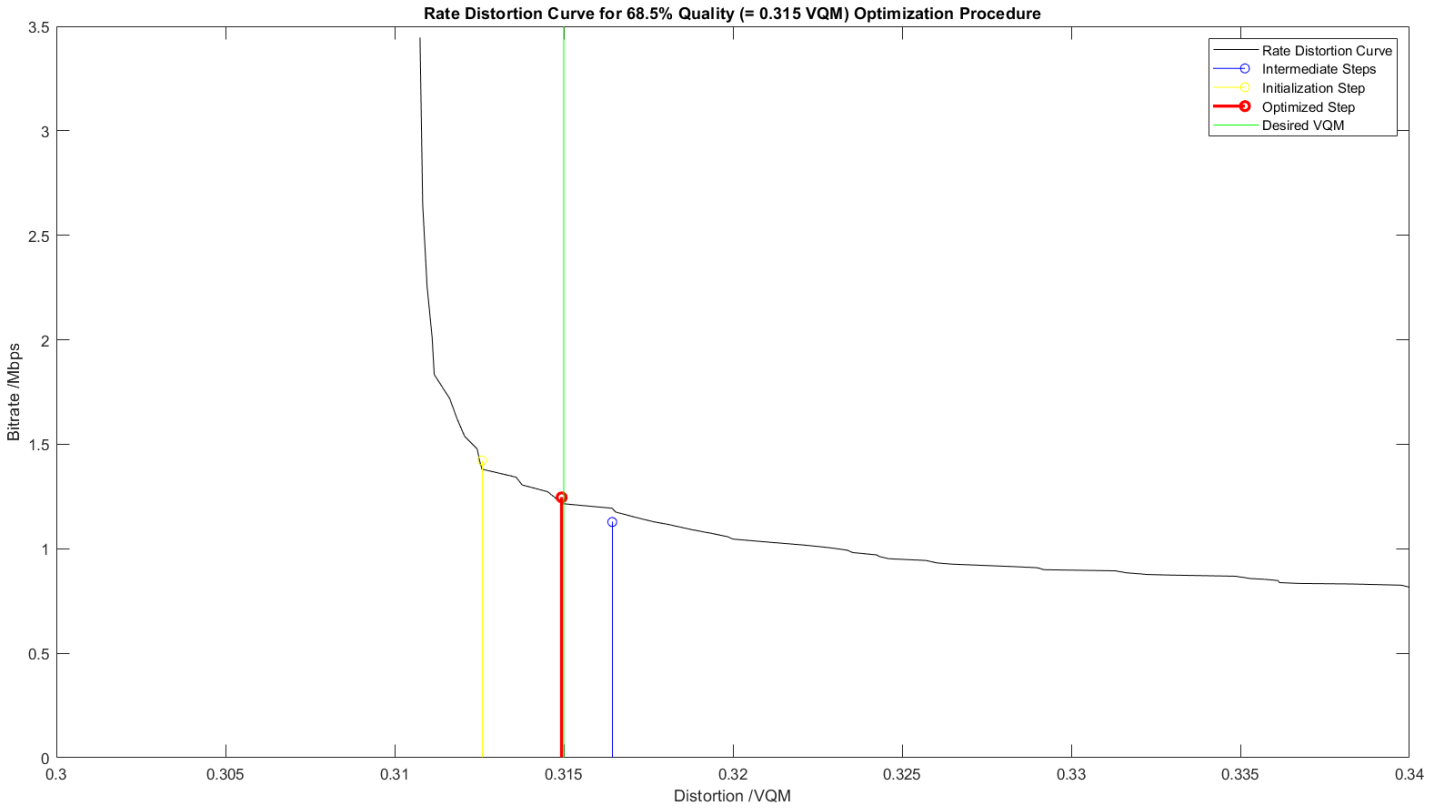
Eg 02:

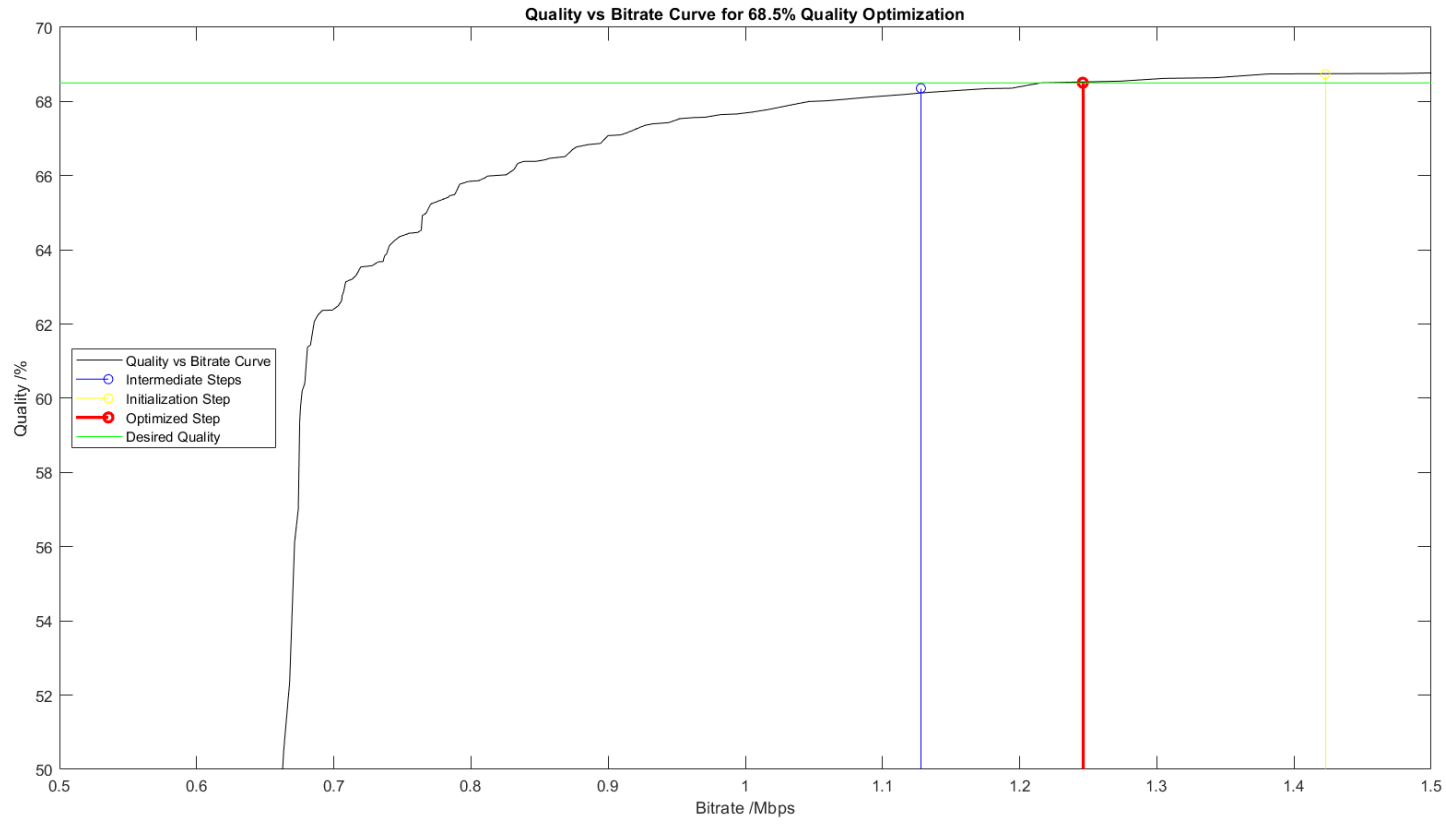
Fixed Quality: 68.5% (=0.315 VQM)

Optimized bitrate: 1.246 Mbps

Quality at optimized level: 68.506% (Error = +0.006%)

Quantization level: 1.5





REFERENCES

[1] Measurement of Objective Video Quality in Social Cloud Based on Reference Metric

(https:/www.hindawi.com/journals/wcmc/2020/5028132/)

[2] VideoLAN Organization (https://www.videolan.org/index.html)