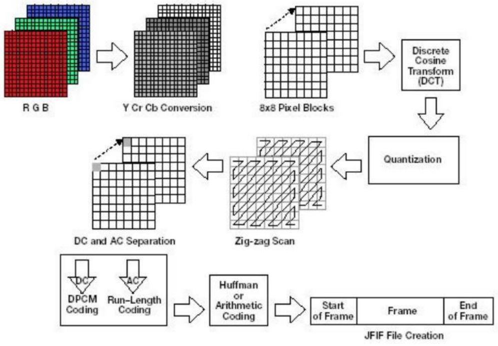
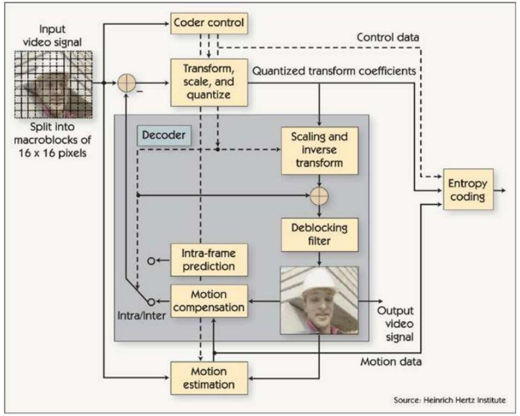
H.264 (MPEG-4 AVC) & JPEG Based Compression & Transmission System

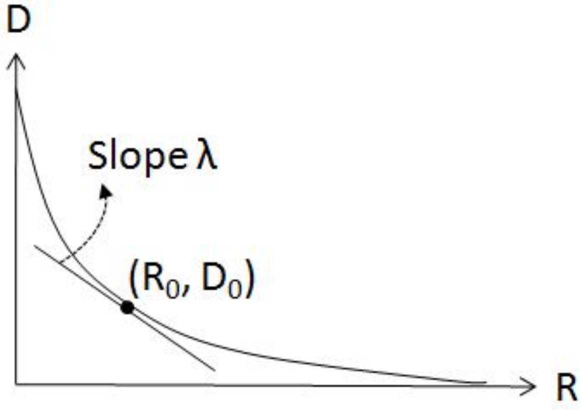
L.B.I.P. Thilakasiri - E/16/367

Objective

- To investigate the basic functions of an image coding system.
- To investigate the basic functions of a block-based video coding system.
- To investigate the Rate -Distortion optimisation techniques used in video coding.

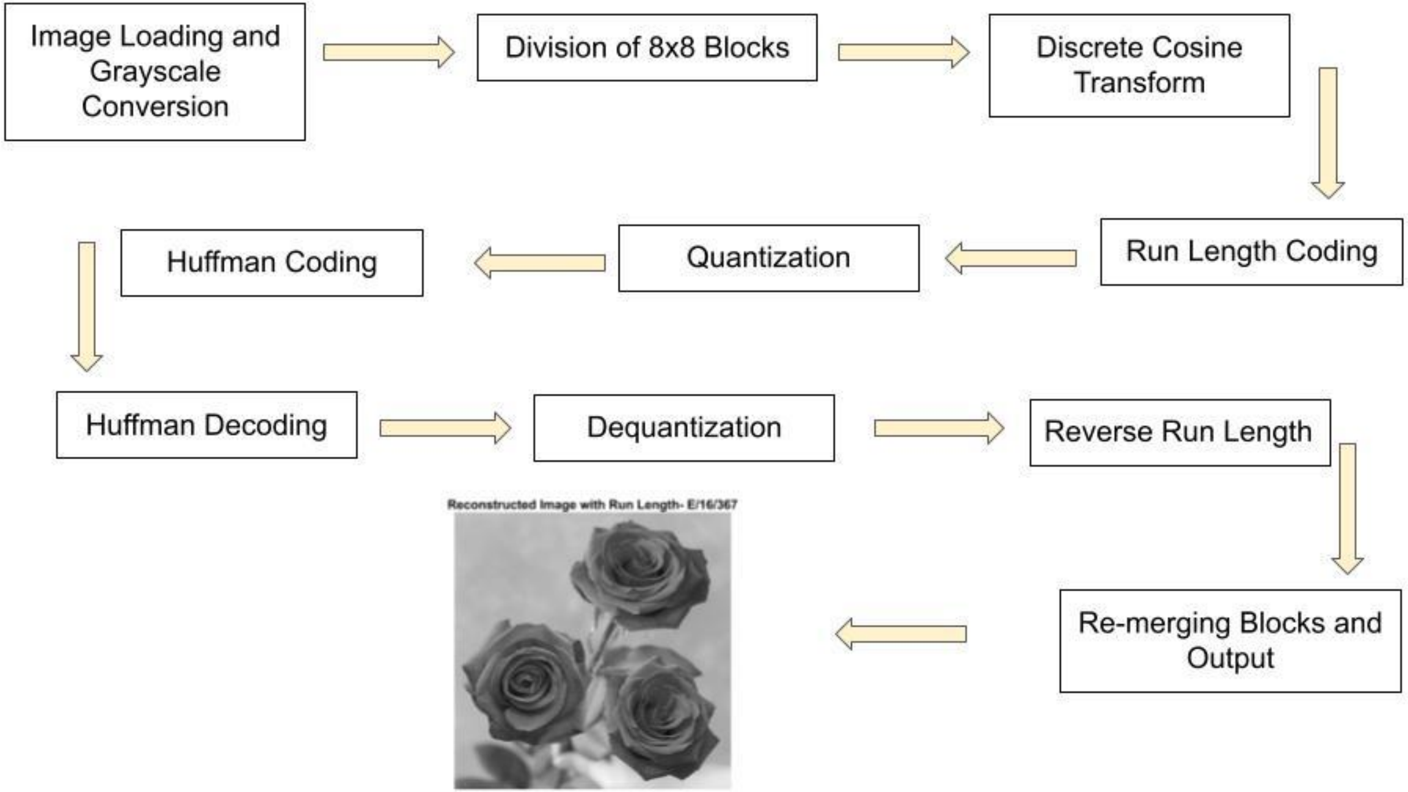






Matlab Implementation

Stage 1: Image Compression



Files Created

- Four files were created
    - main.m
    - mainWithRunLength.m
    - run_length_coding.m
    - Run_length_decoding.m

Available Quantization Methods

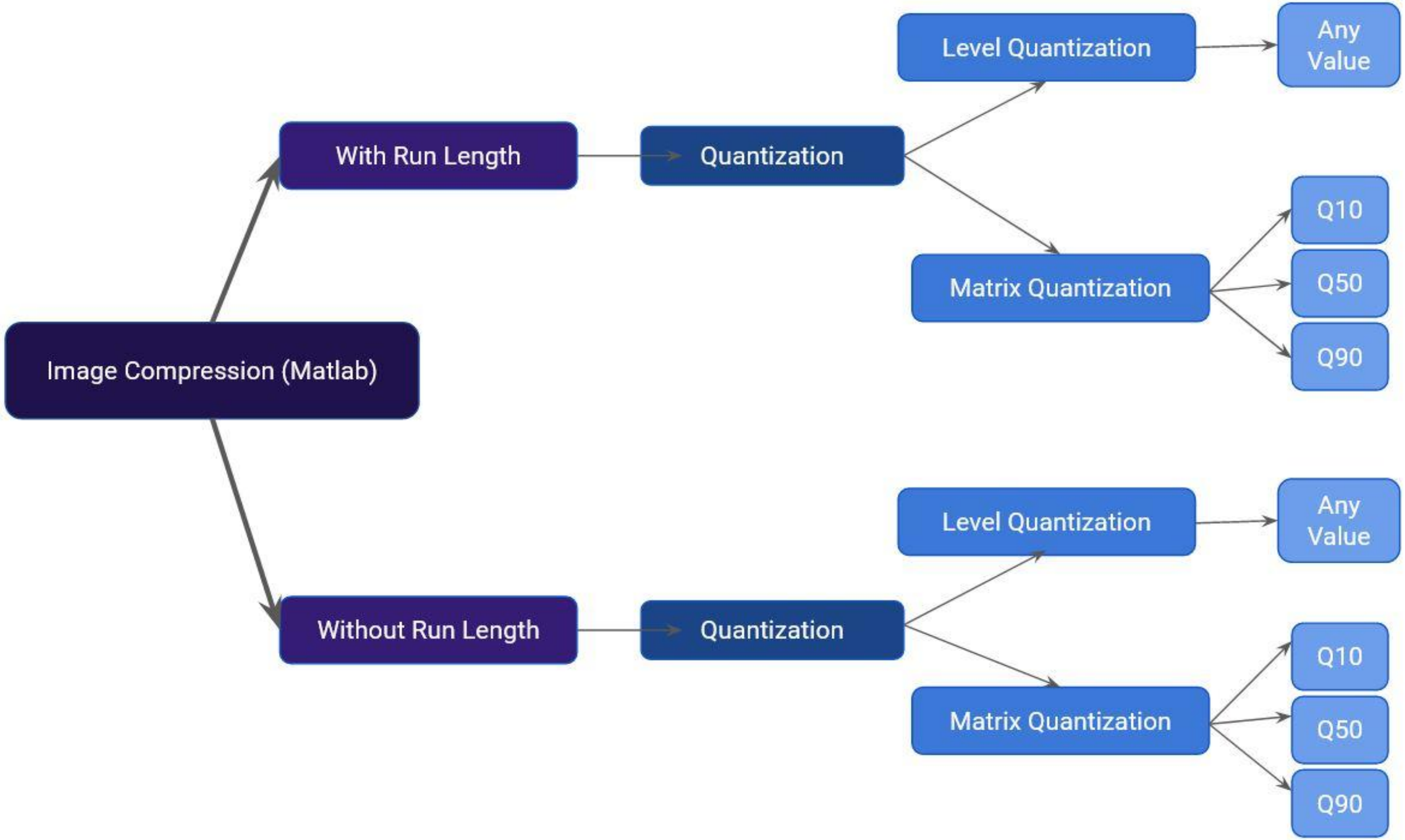


Image Used

Original Image - E/16/367

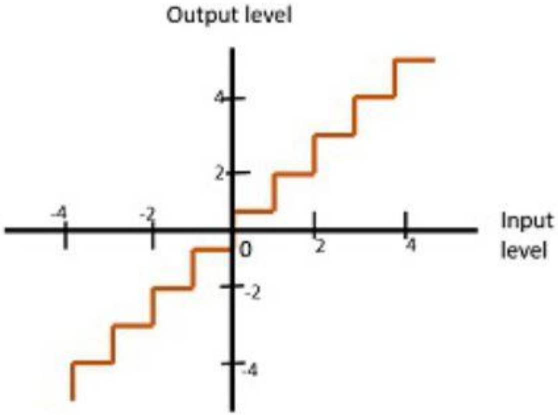
Original Image in GrayScale - E/16/367

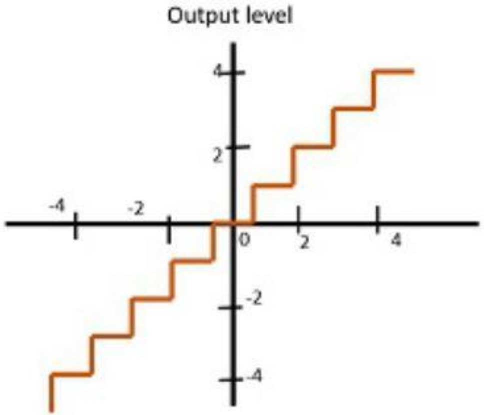
User Input

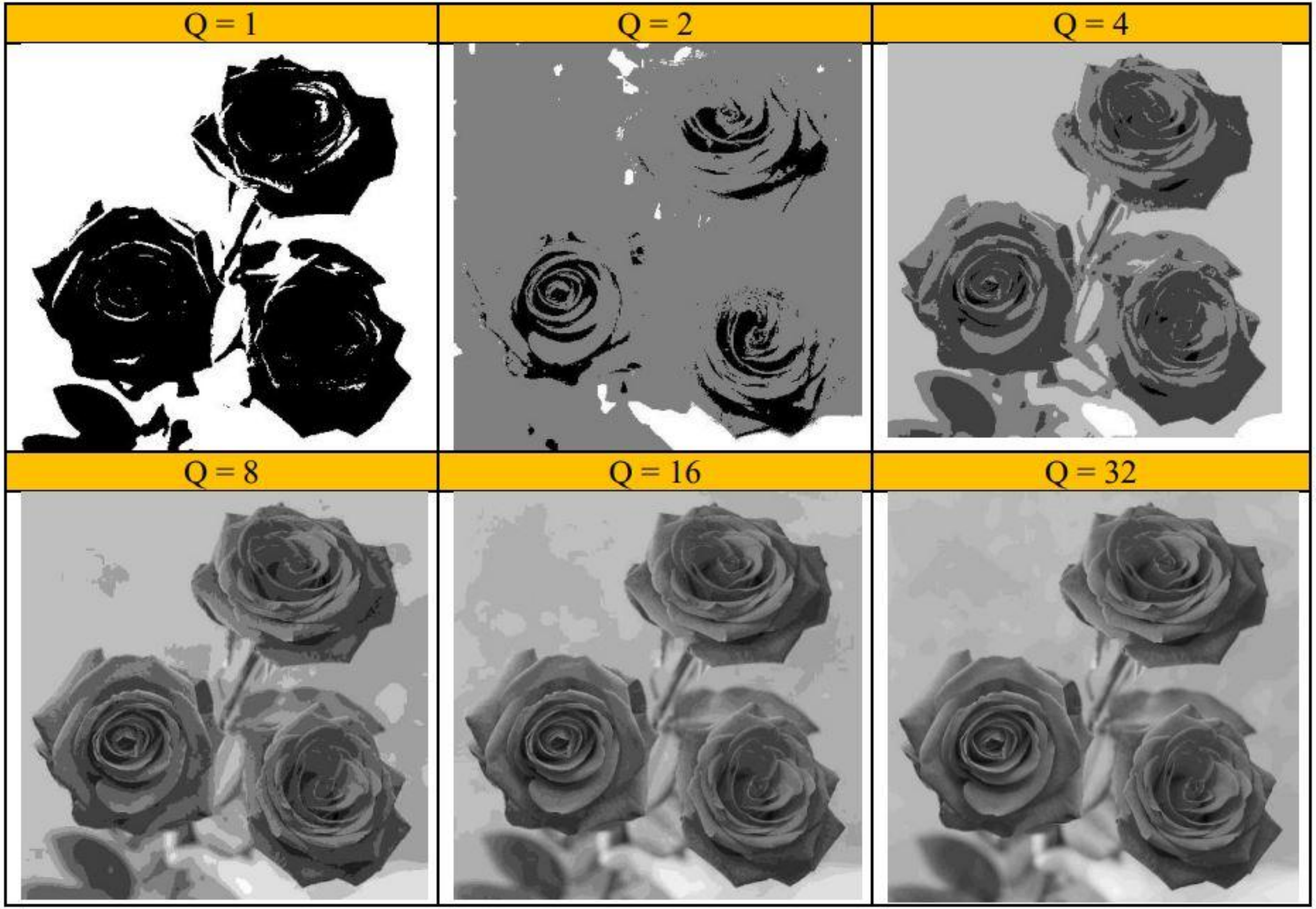
```
Command Window

Requested Quantization Method => Level Quantization (1), JPEG Matrix (2) ? 1
Requested Quantization Level ? 1

Q =

     1
```

Quantization by Levels

- Can use any number of levels
- Built in function imquant was not used






Q = 1
Q = 2
Q = 4
Q = 8
Q = 16
Q = 32

Quantization by Matrix

- Q10, Q50, Q90 - (10%, 50%, 90% of the original image)

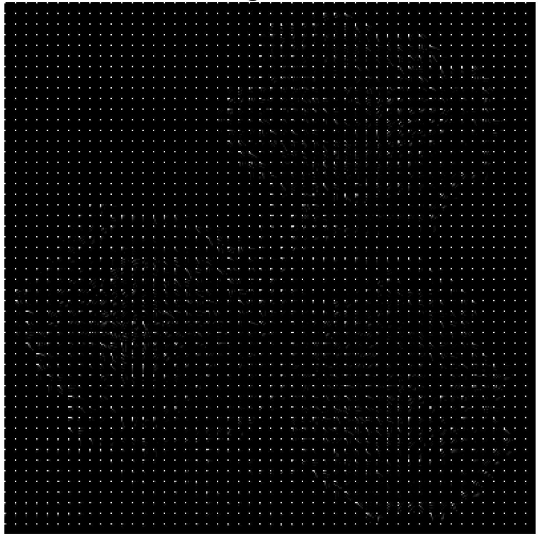
```
Q10 = ([[80,60,50,80,120,200,255,255],
       [55,60,70,95,130,255,255,255],
       [70,65,80,120,200,255,255,255],
       [70,85,110,145,255,255,255,255],
       [90,110,185,255,255,255,255,255],
       [120,175,255,255,255,255,255,255]
       [245,255,255,255,255,255,255,255]
       [255,255,255,255,255,255,255,255]
```

```
Q50 = ([[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],
       [72,92,95,98,112,100,130,99]])
```

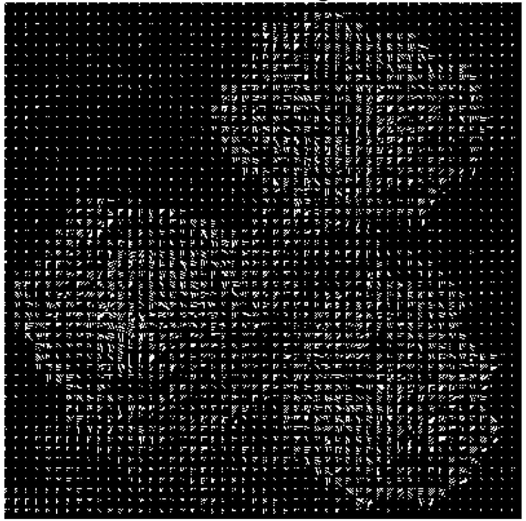
```
Q90 = ([[3,2,2,3,5,8,10,12],
       [2,2,3,4,5,12,12,11],
       [3,3,3,5,8,11,14,11],
       [3,3,4,6,10,17,16,12],
       [4,4,7,11,14,22,21,15],
       [5,7,11,13,16,12,23,18],
       [10,13,16,17,21,24,24,21],
       [14,18,19,20,22,20,20,20]])
```

Results

DCT Image - E/16/367

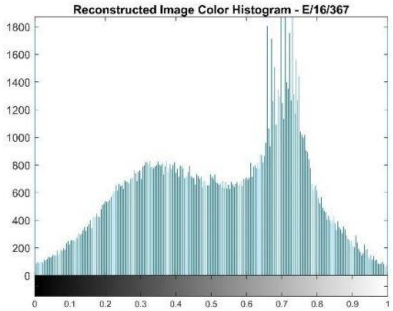
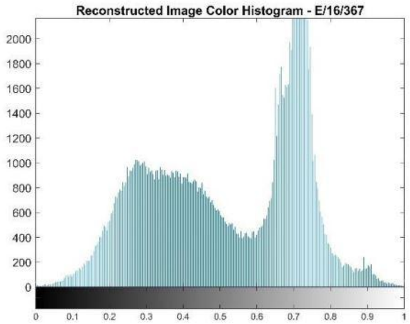
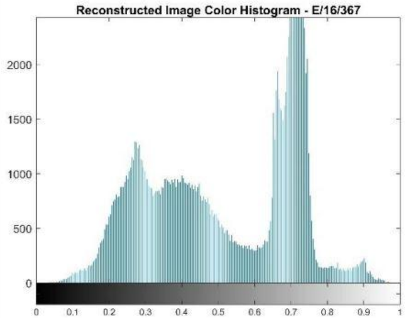


Huffman Decoded Image - E/16/367



Reconstructed Image - E/16/367

Results

Q10	Q50	Q90
		
		

Stage 2: Video Compression

Reading the Frames and GrayScale Conversion

Division of 8x8 Blocks of the Previous Frame

Discrete Cosine Transform of the Previous Frame

Quantization of the Previous Frame

Huffman Coding of the Previous Frame

Huffman Decoding of the Previous Frame

Dequantization of the Previous Frame & Re-merging Blocks

Store the Predicted Lossy Ref Frame Based on Original Previous Frame

Motion Estimation, Motion Vectors and Block Matching for the Incoming Frames

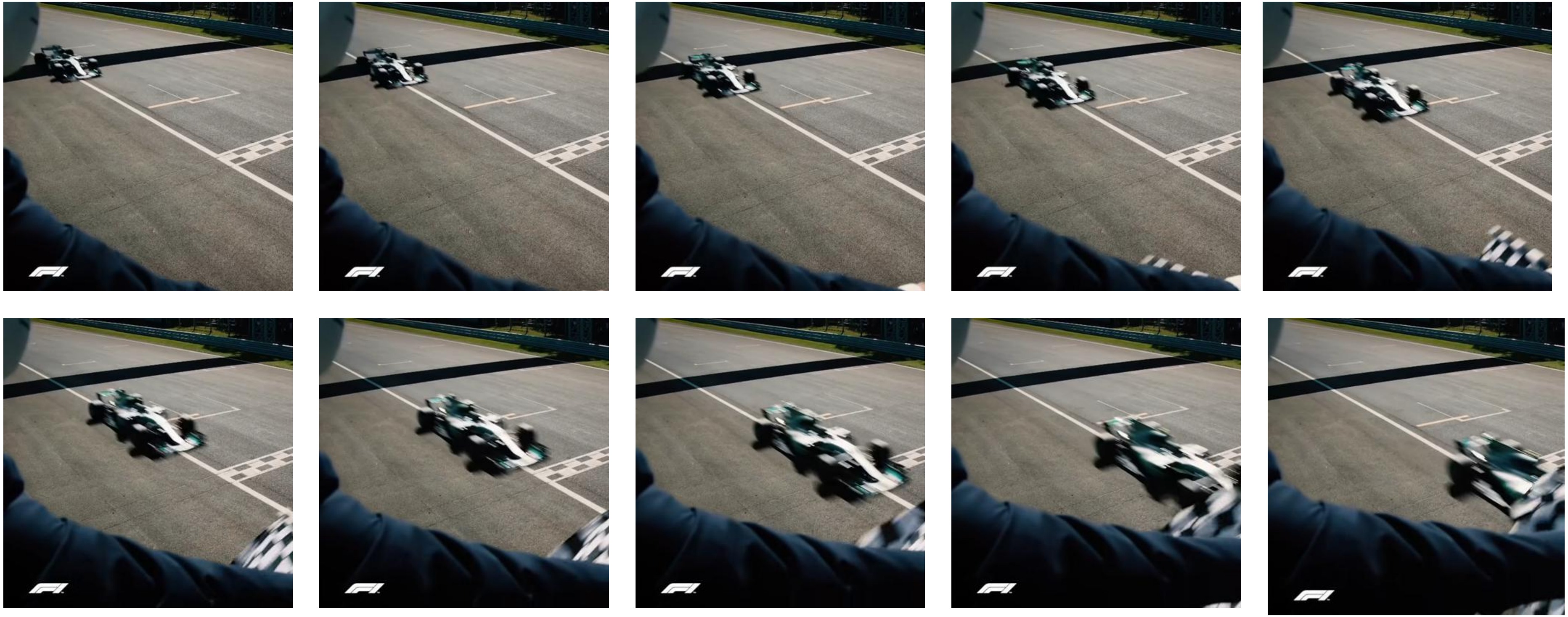
Transmission

Recreating the Frames using the MV, Ref and Residues

Files Created

- Ten files were created
  - main_vid.m
  - motionVec_and_Residue.m
  - SAD_Calc.m
  - Quantization.m
  - motion_estimation.m
  - huffman_encode.m
  - huffman_decode.m
  - deQuantization.m
  - compensated_img.m
  - img_transmission_process.m

Frames Used



User Input

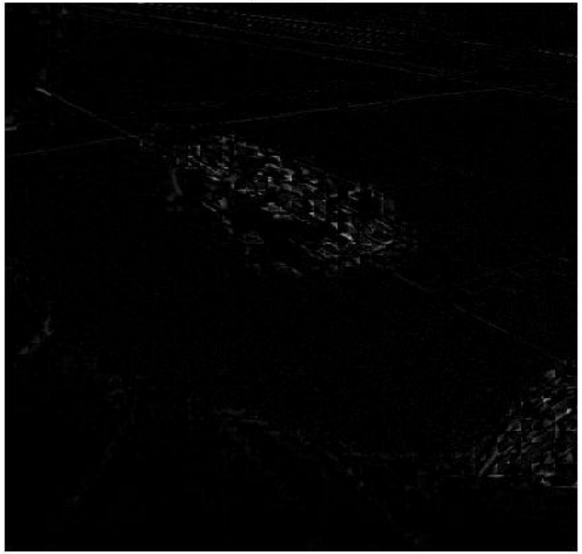
Command Window

```
Requested Quantization Method => Level Quantization (1), JPEG Matrix (2) ? 1
Requested Quantization Level ? 1

Q =

     1
```

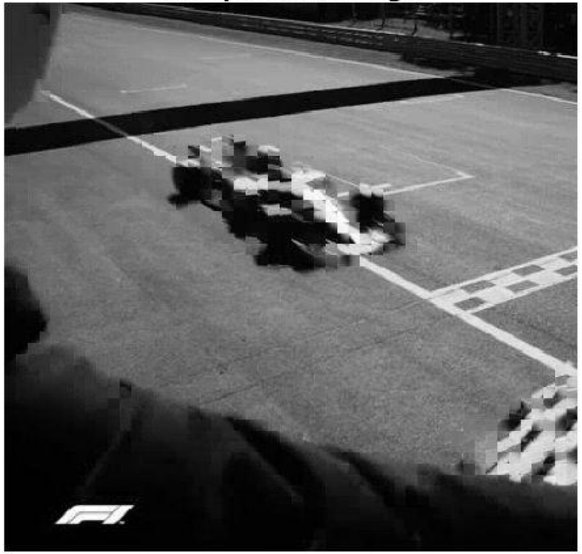
Results

- Example Output for the 7th Frame

Residue

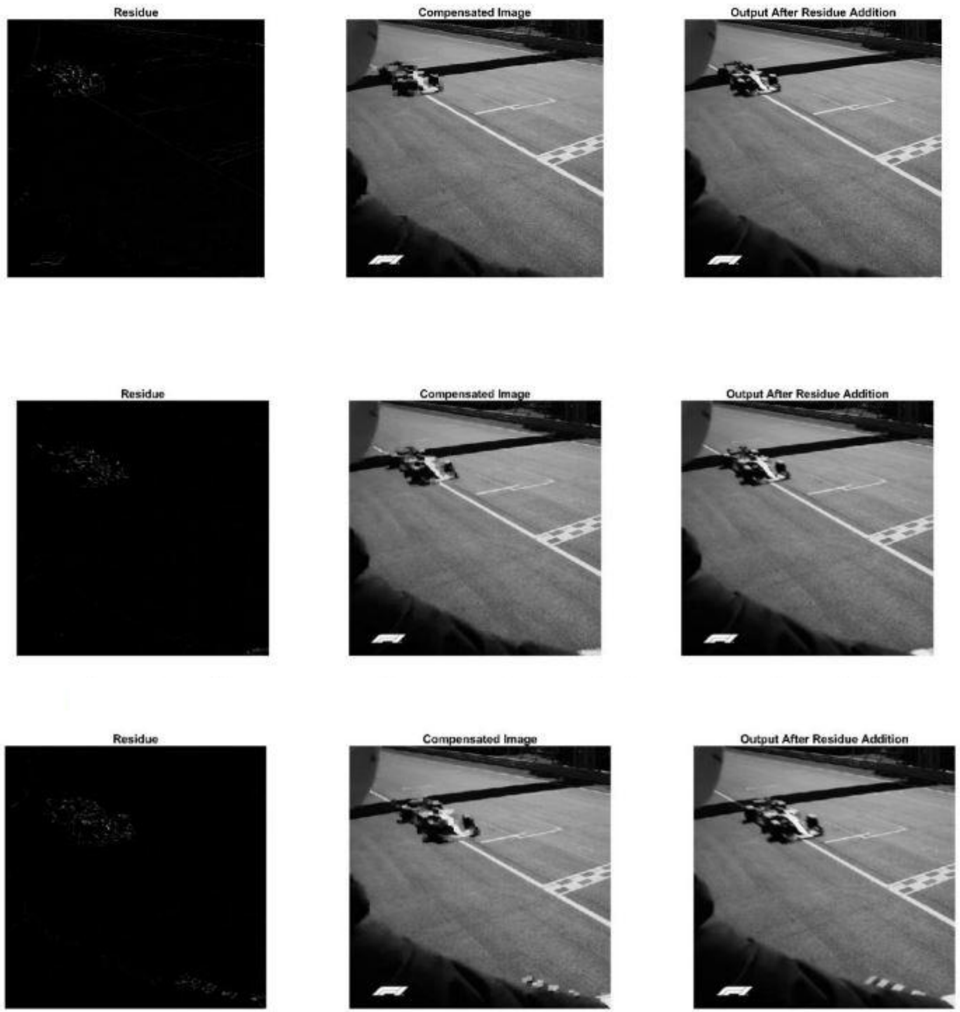


Compensated Image

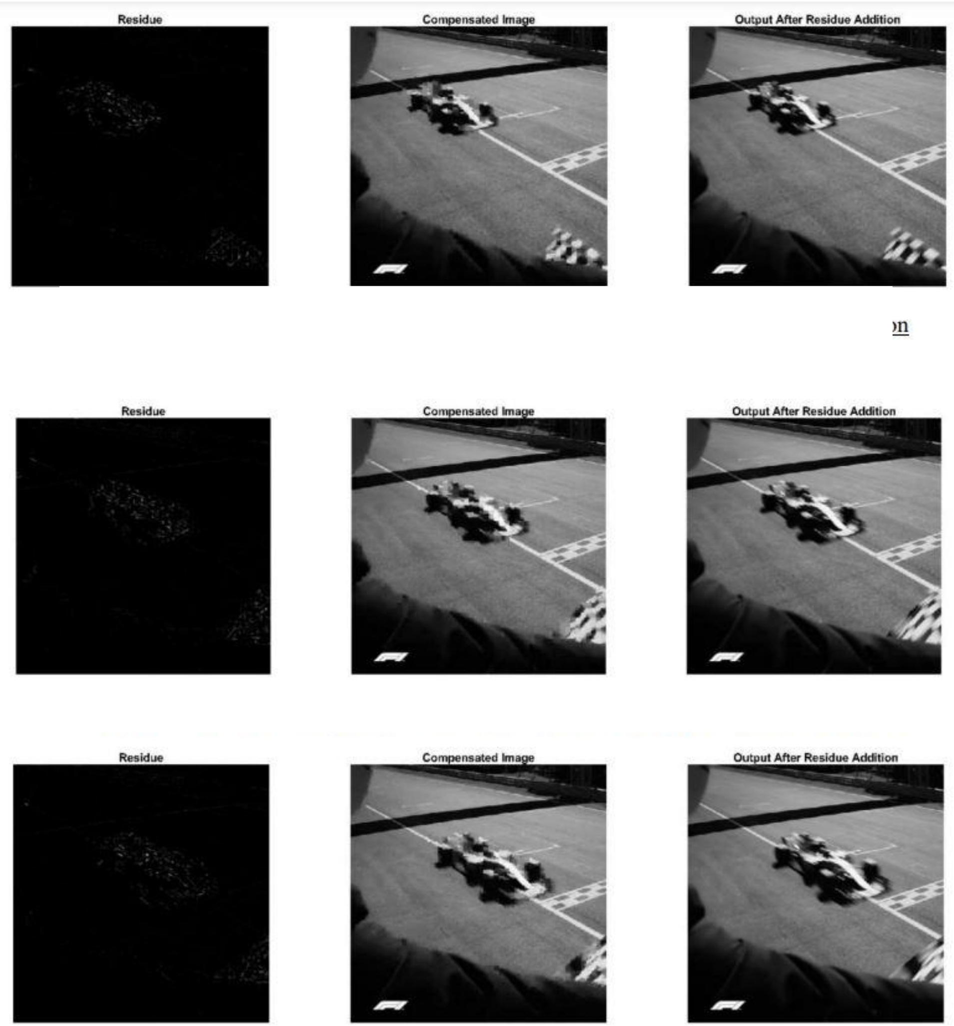


Output After Residue Addition

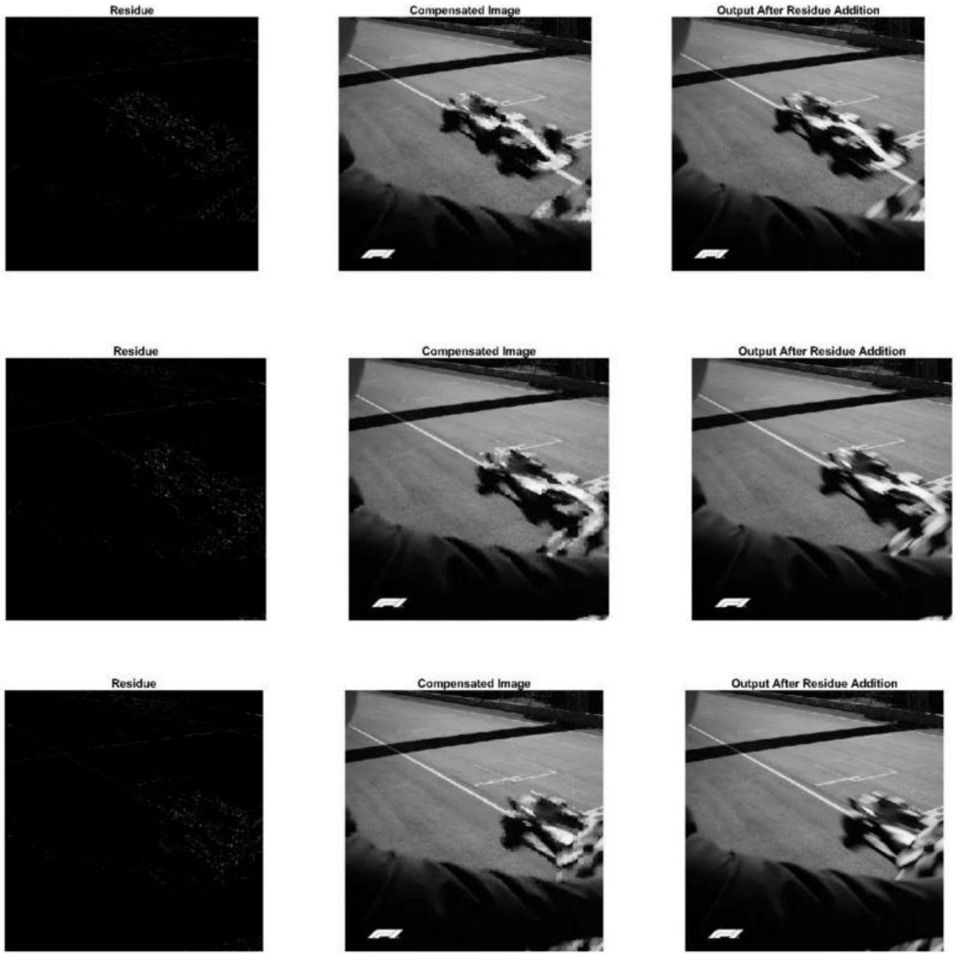
Results


Residue
Compensated Image
Output After Residue Addition
Residue
Compensated Image
Output After Residue Addition
Residue
Compensated Image
Output After Residue Addition

Results

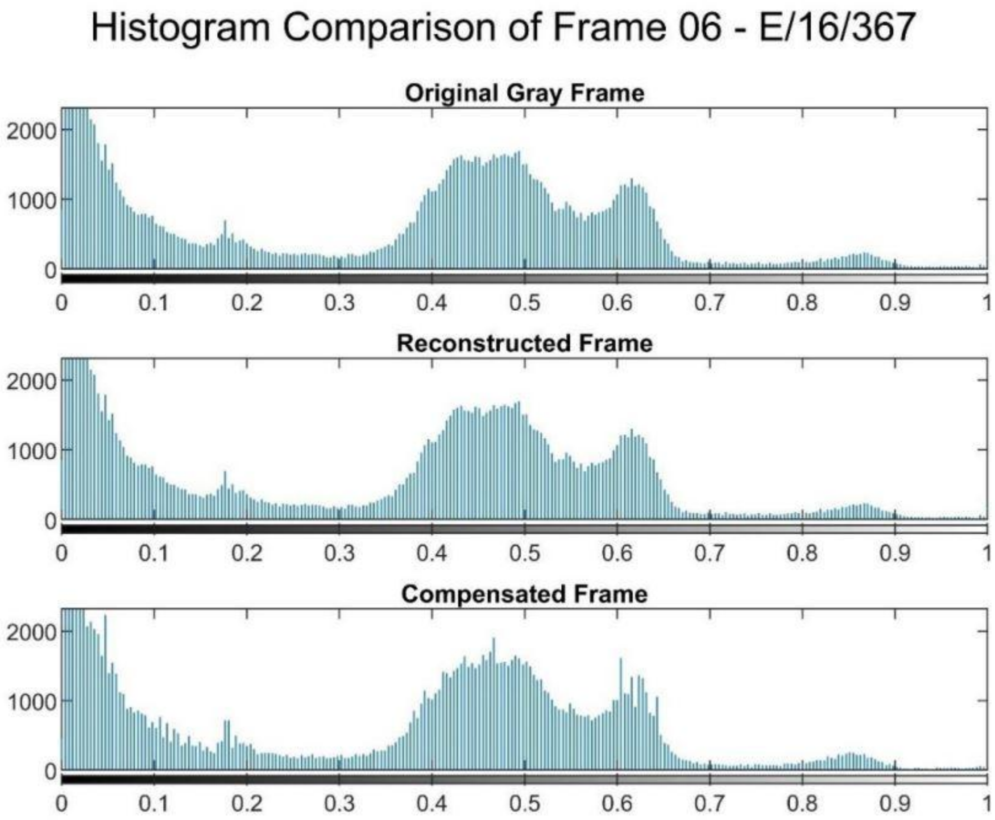

Residue
Compensated Image
Output After Residue Addition
Residue
Compensated Image
Output After Residue Addition
Residue
Compensated Image
Output After Residue Addition

Results


Residue
Compensated Image
Output After Residue Addition
Residue
Compensated Image
Output After Residue Addition
Residue
Compensated Image
Output After Residue Addition

Results

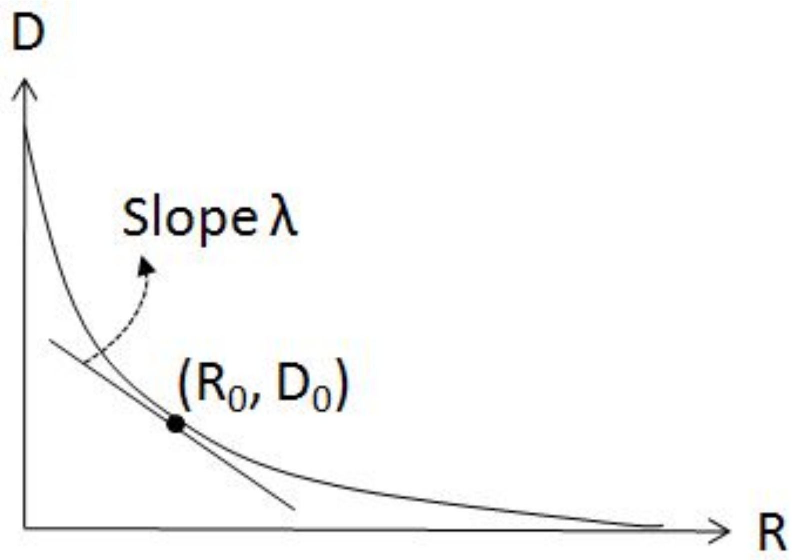
- Histograms for all frames is given in the report.
- Reconstructed images are vastly different from the original only at Quantization by levels.



Stage 3: Improved Hybrid Video Codec

Files Created

- One file was created
  - quality.m



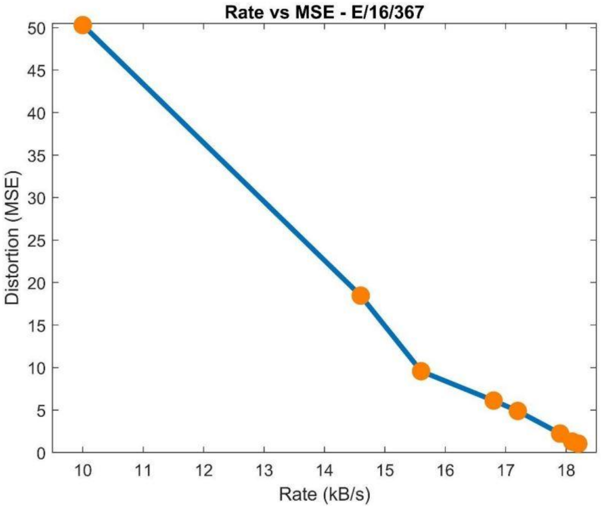
Results

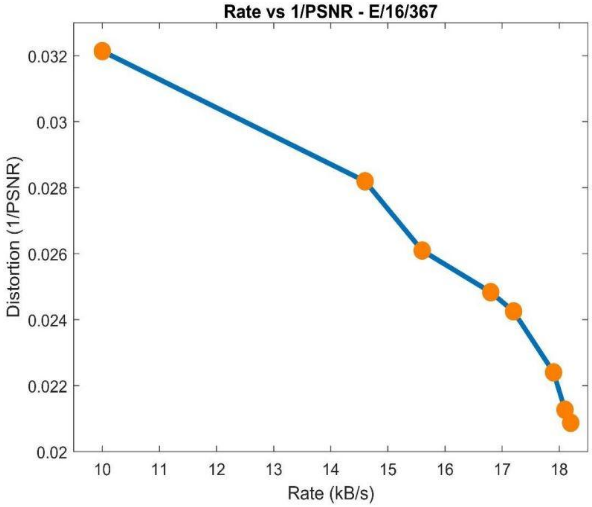
Quantization level	File Size (kB)	MSE
20	18.2	1.0542
16	18.1	1.2936
8	17.9	2.2359
5	17.2	4.8946
4	16.8	6.1141
3	15.6	9.5738
2	14.6	18.4629
1	10.0	50.2919

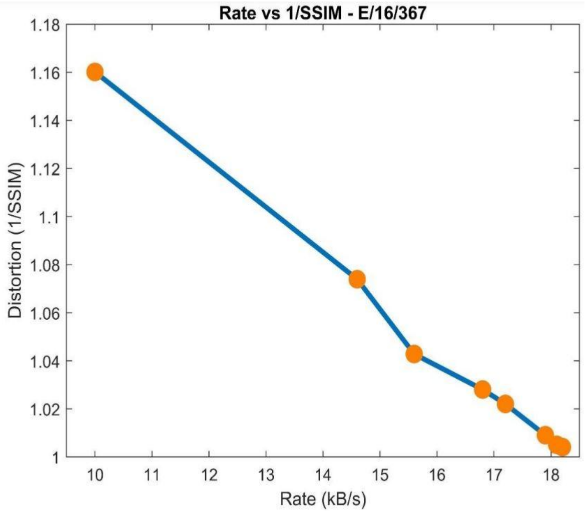
Quantization level	File Size (kB)	PSNR	Distortion (1/PSNR)
20	18.2	47.9016	0.0209
16	18.1	47.0127	0.0213
8	17.9	44.6363	0.0224
5	17.2	41.2337	0.0243
4	16.8	40.2675	0.0248
3	15.6	38.3200	0.0261
2	14.6	35.4678	0.0282
1	10.0	31.1158	0.0321

Quantization level	File Size (kB)	SSIM	Distortion (1/SSIM)
20	18.2	0.9959	1.0041
16	18.1	0.9950	1.0051
8	17.9	0.9910	1.0091
5	17.2	0.9784	1.0221
4	16.8	0.9727	1.0281
3	15.6	0.9589	1.0428
2	14.6	0.9312	1.0739
1	10.0	0.8620	1.1601

Results



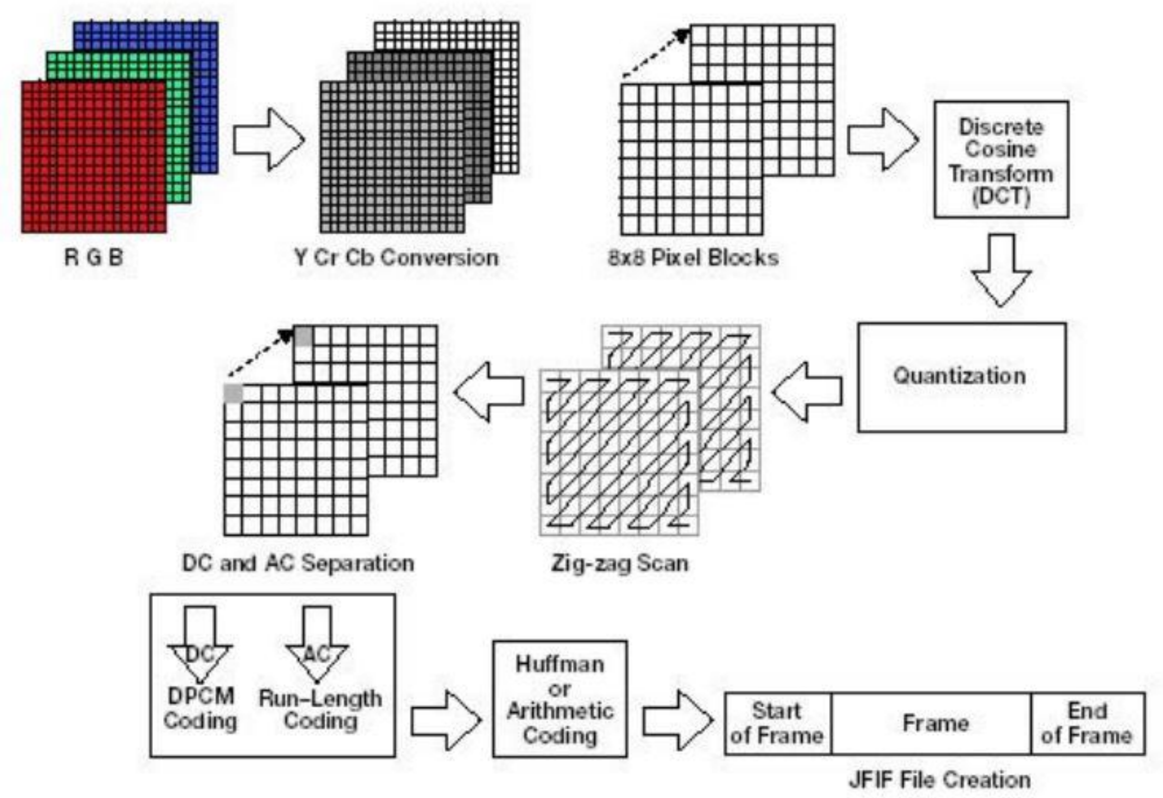




- Can be concluded that the 15.5 kB/s rate is the best approach for optimal transmission.

Python Implementation

Python Implementation



Libraries Used

```
import cv2
from skimage.io import imread
from skimage.color import rgb2gray
import numpy as np
import matplotlib.pylab as plt
from scipy.fftpack import dct, idct
import PIL
from PIL import Image
import time
import collections
import datetime
import re
```

8x8 Window Blocking

```
def block_Div(path):

    row = 8
    col = 8
    windowed = []
    img = imread(path)
    #gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    for r in range(0,img.shape[0] - row, row):
        for c in range(0,img.shape[0] - col, col):
            windowed.append(img[r:r+row,c:c+col])
    return windowed
```

Discrete Cosine Transform

```
def DCT(block_array):

    imF = []
    for i in range (len(block_array)):
        imF.append(dct(dct(block_array[i].T, norm='ortho').T, norm='ortho'))

    imF = np.around(imF)

    return imF
```

Quantization

```
def quantization(array):

    level = int(input("Enter the Required Quantization Level (1 or 2 or 3):"))

    l1_norm = np.array([[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]])
    quantized = []
    quant = [[]]
    quant = np.array(quant)
    if level == 1:
        for i in range (len(array)):
            quantized.append(np.array(array[i])/l1_norm)
            #print(array[i]/l1_norm)
##      for j in range (len(quantized)):
##          quant =  np.concatenate((quant, quantized[i]), axis=0)
    quantized = np.around(quantized)
    print('Quantized:', quantized)

    #np.concatenate((a, b), axis=0)
##              array = Image.fromarray(array)
##              array = array.convert("L")
##              array.save('DCTBeforeQuantizing.jpg')
##              quantized = array.quantize(level)
##              quantized = quantized.convert("L")
##              quantized.save('DCTAfterQuantizing.jpg')

    return quantized
```

ZigZag Code

File Edit Format Run Options Window Help

```
import numpy as np

def zigzag(AC):
    AC_list = []
    for i in range (len(AC)):
        AC_list.append((AC[i]).tolist())
    AC_list = np.array(AC_list)
    #AC_list = np.array(list(map(int, AC_list)))
    print(type(AC_list))
    print(AC)
    AC_row = np.concatenate([np.diagonal(AC_list[::-1,:],
        1+i)[::(2*(i % 2)-1)] for i in range(1-AC_list.shape[0], AC_list.shape[0])])
    #print('acrow:',AC_row)
    AC_row = AC_row.tolist()
    AC_row = list(map(int, AC_row))
    AC_row.insert(0,int(AC_list[0][0]))
    #print('acrow:',AC_row)

    return AC_row


bb = [[1,2,3],[4,5,6],[7,8,9]]
b = zigzag(np.array(bb))
print('ZigZag BitStream:', b)
```

```
<class 'numpy.ndarray'>
[[1 2 3]
 [4 5 6]
 [7 8 9]]
ZigZag BitStream: [1, 2, 4, 7, 5, 3, 6, 8, 9]
>>>
```

DC Separation, Difference Coding & Reverse Difference Coding

File Edit Format Run Options Window Help

```
def reverse_differential(dc_decoded):
    dc_rev = []
    for i in range (len(dc_decoded)):
        #print(dc_rev)
        if i == 0:
            dc_rev.append(dc_decoded[i])
        else:
            #print(dc_rev[i-1],dc_decoded[i])
            dc_rev.append(dc_rev[i-1] - dc_decoded[i])

    return dc_rev

def differential_coding (quantized):
    DC = []
    print('Recieved DC BitStream: ',quantized)
    for i in range (len(quantized)):
        if i == 0:
            DC.append(quantized[i])
        else:
            DC.append(quantized[i-1] - quantized[i])
    #DC = list(map(int, DC))
    print('Difference Coded DC BitStream:',DC)
    return DC

dc = [1,2,3,4,5,6,7,8,9]
c = differential_coding(dc)
x = print('Final:',reverse_differential(c))
```

```
Recieved DC BitStream:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
Difference Coded DC BitStream: [1, -1, -1, -1, -1, -1, -1, -1, -1]
Final: [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

AC-DC Combine

```python
File  Edit  Format  Run  Options  Window  Help
def ac_dc_combine(ac_rev, dc_rev):

    arr_new = []
    print('Recieved AC BitStream:',ac_rev,'\n')
    print('Recieved DC BitStream:',dc_rev,'\n')
    #arr_new = np.array(arr_new)
    for i in range (len(dc_rev)):
        temp1 = []
        for j in range(8):
            temp2 = []
            temp2.append(dc_rev[i])
            for k in range (7):
                temp2.append(ac_rev[i*8*8+k])
            temp1.append(temp2)
        arr_new.append(temp1)
        #print(arr_new[i],'\n')
    #print(arr_new)

    return(arr_new)


dc_rev = [1,2,3,4,5]
ac_rev = []
ac_rev = ac_rev + [1]*63 + [2]*63 + [3]*63 + [4]*63 + [5]*63
arr_new = ac_dc_combine(ac_rev, dc_rev)
print('Combined DC-AC BitStream:',arr_new,'\n')
```

```
Recieved AC BitStream: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4
, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
, 5, 5, 5]

Recieved DC BitStream: [1, 2, 3, 4, 5]

Combined DC-AC BitStream: [[[1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1], [1
, 1, 1, 1, 1, 1, 1]], [[2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2
, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2,
2, 2, 2, 2, 2, 2, 2]], [[3, 3, 3, 3, 3, 3, 3, 3], [3, 3, 3, 3, 3, 3, 3, 3], [3, 3, 3, 3, 3, 3, 3, 3], [3,
3, 3, 3, 3, 3, 3, 3], [3, 3, 3, 3, 3, 3, 3, 3], [3, 3, 3, 3, 3, 3, 3, 3], [3, 3, 3, 3, 3, 3, 3, 3], [3, 3,
3, 3, 3, 3, 3, 3]], [[4, 4, 4, 4, 4, 4, 4, 4], [4, 4, 4, 4, 4, 4, 4, 4], [4, 4, 4, 4, 4, 4, 4, 4], [4, 4,
4, 4, 4, 4, 4, 4], [4, 4, 4, 4, 4, 4, 4, 4], [4, 4, 4, 4, 4, 4, 4, 4], [4, 4, 4, 4, 4, 4, 4, 4], [4, 4, 4,
4, 4, 4, 4, 4]], [[5, 5, 5, 5, 5, 5, 5, 5], [5, 5, 5, 5, 5, 5, 5, 5], [5, 5, 5, 5, 5, 5, 5, 5], [5, 5, 5,
5, 5, 5, 5, 5], [5, 5, 5, 5, 5, 5, 5, 5], [5, 5, 5, 5, 5, 5, 5, 5], [5, 5, 5, 5, 5, 5, 5, 5], [5, 5, 5, 5,
5, 5, 5, 5]]]

>>>
```

IDCT

```python
def IDCT(path):
    img = np.asarray(Image.open(path),np.uint8)
    im1 = idct(idct(img.T, norm='ortho').T, norm='ortho')
    im1 = Image.fromarray(im1)
    im1 = im1.convert("L")
    im1.save('Output.jpg')
    return None
```