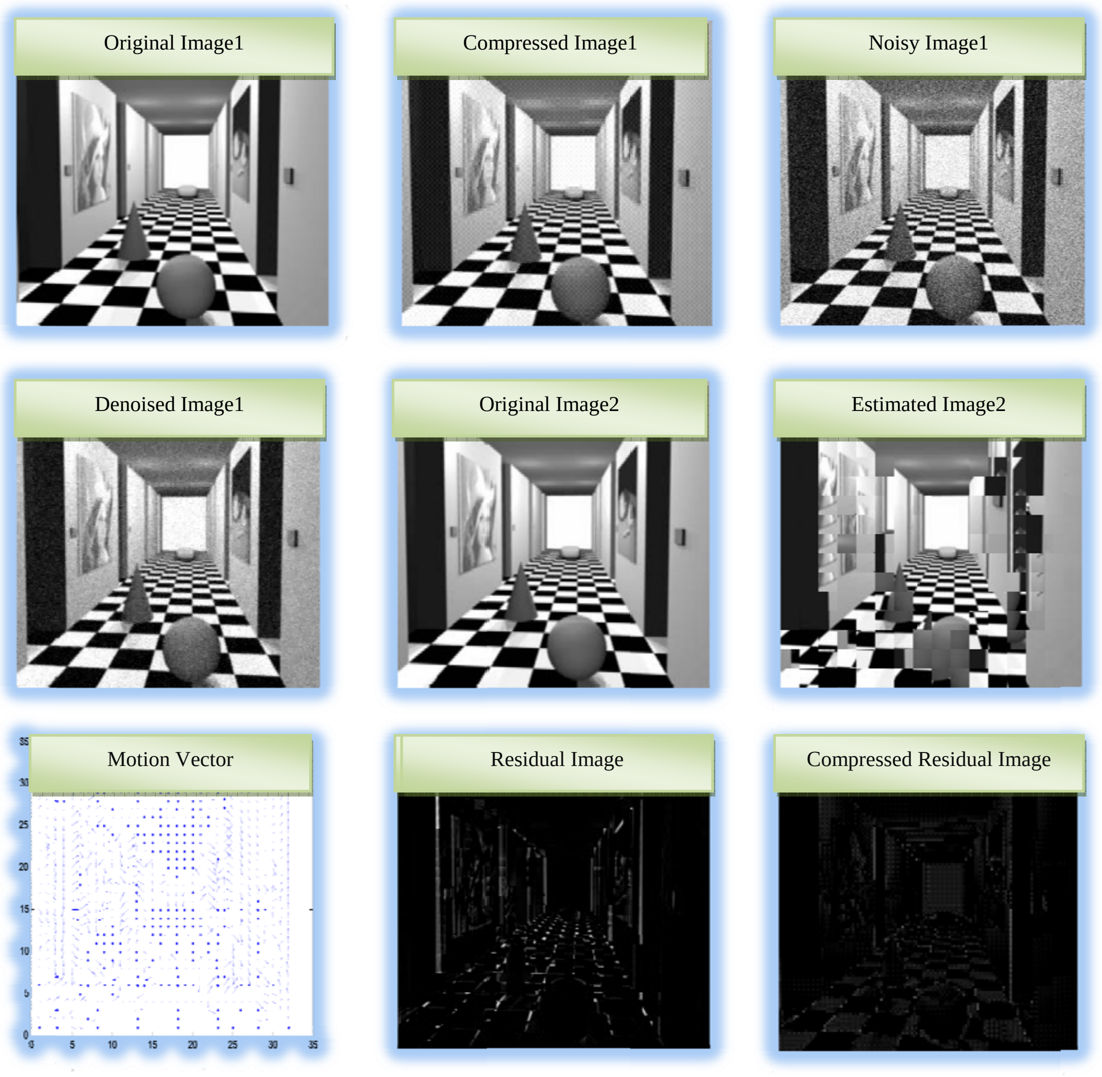
Imperial College London

Department of Electrical and Electronic Engineering
Final Year Project Report 2007

Title: # Distributed Image Compression

Name: Hojjat Akhondi Asl

Course: 4T



Project Supervisor: Dr. Pier Luigi Dragotti
Second Marker: Mr. Mike Brookes

Abstract

The emerging wireless camera networks have restrictions on battery life, memory and computational power. The wireless cameras cannot communicate with each other due to power constraints and inter-source communication costs. The problem of resource constraint wireless camera network can be alleviated with the use of distributed image coding. The idea of distributed image coding is used to compress images acquired from the camera network by exploiting the statistical properties between the captured images. The aim of this project is to see how the distributed coding system performs when compared to Joint source coding and Independent coding with the use of wavelet transform. We model our camera network with a pair of stereo pictures. For side information, we allow the cameras to overhear a coded and noisy version of what the first camera is sending to the decoder.

Several design issues and implementations of the algorithm are discussed. The bit-rate that the reference image is compressed at, the variance of the channel noise and the bit allocation process, play an important part on how our algorithm performs. It can be said that although being as efficient as joint source coding with distributed coding is practically impossible, but it is possible to get very close to it.

Table of Contents

Table of Figures

Acknowledgements

First of all, I want to thank my parents and my brothers Alireza and Hossein for their support, encouragement and understanding.

I would also like to thank my project supervisor Dr. Luigi Dragotti for his guidance.

Chapter 1

Introduction and Background

Transmission of uncompressed audio, graphics and video data requires considerable amount of transmission bandwidth, data storage, power, processing speed and transmission time. It is possible to reduce the size of the multimedia signals by removing the redundancies and irrelevancies. Redundancy reduction aims at removing duplication from the signal and irrelevancy reduction aims at omitting parts of signal that will not be noticed by the human visual system.

Advances in camera technology and wireless communication have enabled the development of low-cost and low-power wireless camera networks which have emerged for variety of reasons including environmental and habitant monitoring, target tracking and surveillance. The main characteristic of such networks is the constraints on resources in terms of energy (battery life), memory and computational power. These resource constraints make the design of wireless camera networks very difficult. The problem of resource constraint wireless camera network can be alleviated with the use of distributed image coding.

Distributed image coding refers to the compression of two or more physically separated sources. These sources, although statistically dependent, cannot communicate with each other due to power constraints and inter-source communication costs. By statistically dependent we mean that the cameras have overlapping fields of view.

In this project we want to use the idea of distributed image coding to compress images acquired from the camera network. The aim of this project is to see how the distributed coding system performs when compared to Joint source coding and Independent coding. We will use wavelet transform to do compression on images captured from the cameras. After distributed compression, these sources send their compressed images to a central point for joint decoding.

1.1 Distributed Source Coding

In order to understand the idea of Distributed source coding, we first need to describe what the terms Joint source coding and Independent coding exactly mean.

Consider a communication system with N sources 1,...,N (see figure 1). The idea is to compress the data from the sources as efficiently as possible at a given bit-rate.

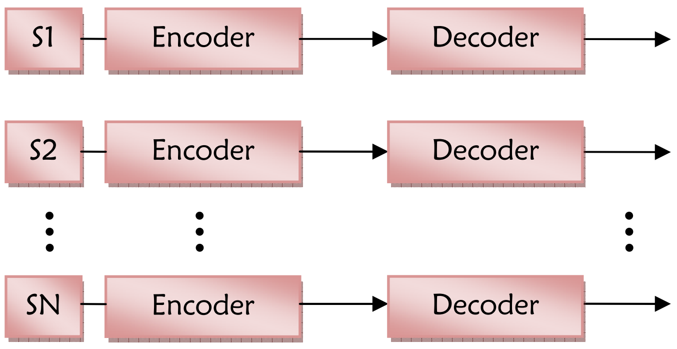


Figure 1 – A communication system with N sources

The simplest approach for encoding the sources is to code them independently of each other in the transmitter and at the receiver we independently decode all the information received from all the sources. Therefore each source has one encoder and one decoder associated with it. This is called Independent Coding and is used when the sources are not correlated with each other.

Now let's assume that the sources are correlated with each other. Since the sources are correlated, we can efficiently code the information from all sources given that any source has direct access to its previous source(s). This is because "source n" needs to send only some of its data to the receiver by knowing the fact that most of its data is in "source n-1". Therefore we can achieve a better image quality or a higher compression rate when compared to the Independent Coding case. This is due to the fact that we only send the residual information from "source n" to the decoder and due to similarities between the adjacent sources the resulting residual image will be small in terms of pixel intensities. Therefore fewer bits is needed for lossless/lossy reconstruction in the decoder than the Independent Coding case, resulting in better qualities at the same bit-rates or smaller bit-rates at the same image qualities. We have to bear in mind that for residual decoding, the decoder needs to have full access to at least one of the sources. This is called Joint Source Coding or JSC for short (see figure 2). For this kind of coding system we only have one encoder and one decoder for all the sources. The Motion-Pictures-Experts-Group (MPEG) uses this kind of compression standard with sources being frames and N being the number of frames from one key frame to another (normally 25).

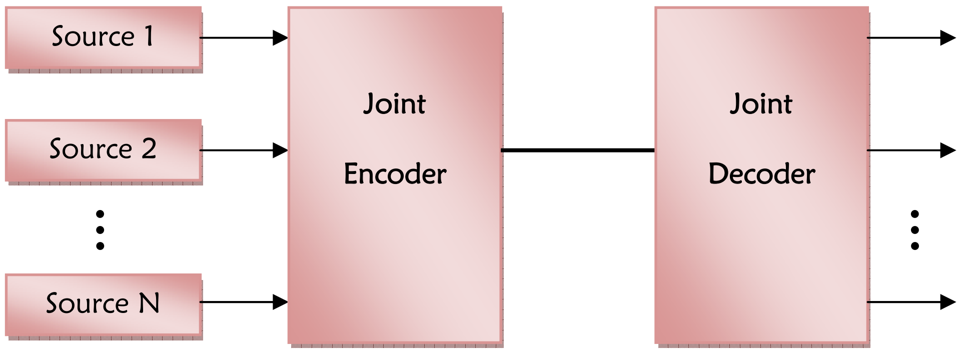


Figure 2 – Joint Source Coding Block Diagram

One disadvantage of this kind of setup is that the sources need to communicate with each other for joint encoding, which in terms of bandwidth and power is costly. This is particularly true for wireless systems. This kind of setup is mainly used for broadcasting or streaming video-on-demand systems where video is compressed once and decoded many times.

Now let's again assume that the source are correlated with each other. Also let's assume that the sources cannot communicate to each other due to power and bandwidth costs. Consider sources 1 and 2 with entropies H(1) and H(2) to be encoded at the rates R_1 and R_2. For the joint source coding case a rate of $R_1 + R_2 \geq H(1,2)$ is sufficient for lossless encoding, where H(1,2) is the joint entropy between the sources 1 and 2. In 1973 Slepian-Wolf [1] proved the surprising result that separate lossless encoding of N correlated discrete sources can be as efficient as joint encoding of the sources assuming that the N compressed sources can be jointly decoded. The theory states that the lossless compression of sources 1 and 2 (separate encoders) is still achievable if:

$R_1 \geq H(1|2)$

$R_2 \geq H(2|1)$ and

$R_1 + R_2 \geq H(1,2)$

Here H(1|2) is the conditional entropy of source 1 given source 2. Thus a combined rate of H(1,2) is sufficient even if the correlated sources are encoded separately as long as they are decoded jointly. However this result assumes that the correlation between the sources is a priori known at each individual encoder.

In 1976, shortly after the Slepian and Wolf's work, Wyner and Ziv [2], [3], [4] extended this work to establish theoretic rate-distortion bounds for lossy compression with the assumption of side information being available at the decoder. Wyner and Ziv said that lossy compression of sources 1 and 2 with independent encoders but joint decoders is as efficient as the lossy compression with joint source coding. Therefore the Wyner- Ziv distributed source coding or Wyner-Ziv coder for short can be thought of quantization (lossy process) followed by a Slepian-Wolf Coder (lossless coder).

This kind of set-up, i.e. distributed source coding system (see figure 3), is mainly used for applications which require low-complexity encoders at the expense of high-complexity decoders Examples are wireless video sensors for surveillance, wireless pc cameras, mobile camera phones and networked cameras [5].

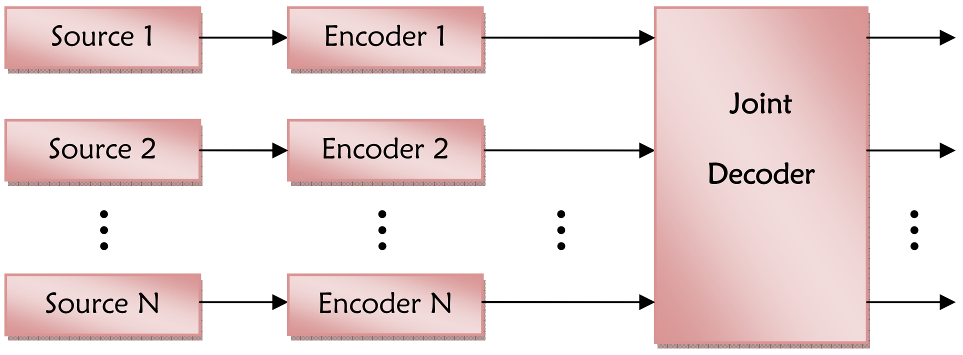


Figure 3 – Distributed Source Coding Block Diagram

Chapter 2

Overview of the algorithm

Consider we have N wireless spatially distributed cameras and we want to compress the images taken from the cameras as efficiently as possible. Due to bandwidth and power costs the cameras are not allowed to communicate to each other therefore each camera has its own encoder. Since all the captured and coded images are sent to the server side we allow joint decoding of the coded images. This is basically the problem of distributed image coding in camera network system.

As mentioned before Slepian-Wolf and Wyner-Ziv proved that compression in a distributed fashion is as efficient as joint coding as long as we have joint decoding in the receiver side. The assumption here is that the statistical properties of each encoder (or the reference encoder) is known to the rest of the encoders.

The aim here is to implement both the distributed image coding and joint image coding systems and see how close the distributed system can get to the joint image coding case. We will also implement the independent coding system so that we can tell how our distributed system is performing overall. The problem that we face here is that how do we make the encoders to know about the statistical properties of the reference encoder? This is a critical assumption to the result of Slepian-Wolf and Wyner-Ziv coders.

2.1 Proposed Solution

First of all we need to make some assumptions about our wireless camera network. We assume that the cameras are identical and spatially distributed. We also assume that the viewed scene by the cameras is static. The scene could also be dynamic but this makes the exploitation of the statistical properties more difficult. All the cameras have the same angle towards the scene therefore we have only horizontal or only vertical shifts for the positions of the cameras. We set the central camera to be the reference source for all the encoders. We let the central encoder to send a complete coded image to the decoder side. This is needed since the central camera is the reference source for all encoders and the decoder needs to have full access to the reference image so that it can reconstruct the rest of the images.

So far whatever we have talked about is really Slepian-Wolf coding. We can do further compression in the encoders by using transform coding. This includes transforming the image to another domain, quantizing (lossy) and Huffman coding which leaves us with a Wyner-Ziv coder. In this project we will use wavelet transform followed by quantization to do compression on the images. Therefore our compression algorithm will be similar to the JPEG2000 codec [6], [7].

Now we need to decide on how to let the cameras to know about the statistical properties of the reference camera. The reference encoder wirelessly sends the coded version of Image1 to the decoder. We can assume that other encoders overhear what the central encoder is sending to the decoder side. By overhearing we mean they can receive a noisy and coded version of Image1. Therefore the encoders need to first denoise and then decode what they are overhearing and then somehow use that to exploit some statistical properties of Image1. By statistical properties we mean the correlation or the similarities among the captured images.

Now the next question is that how do we find the similarities between Image1 and the rest of the captured images [8], [9], [10]? We can use the idea of motion compensation to find an estimated version of the captured image from the reference image (Image1). By doing this we can find the motion vector of the captured Image relative to Image 1. Also the residual image, which is the difference between the captured ImageN and its estimated version, is recorded. The residual image is also transform coded and is sent to the decoder side with the motion vector. Block matching algorithm will be explained in detail in Chapter 3.

In this project, we assume that sensors are two grey-scale stereo image pairs of size 512x512 and we try to develop a distributed image compression algorithm based on the assumptions made and the use of the wavelet transform. Figure 4 shows a rough sketch of the wireless camera network scenario with the implementation of distributed coding technique for images captured with overlapping fields of view.

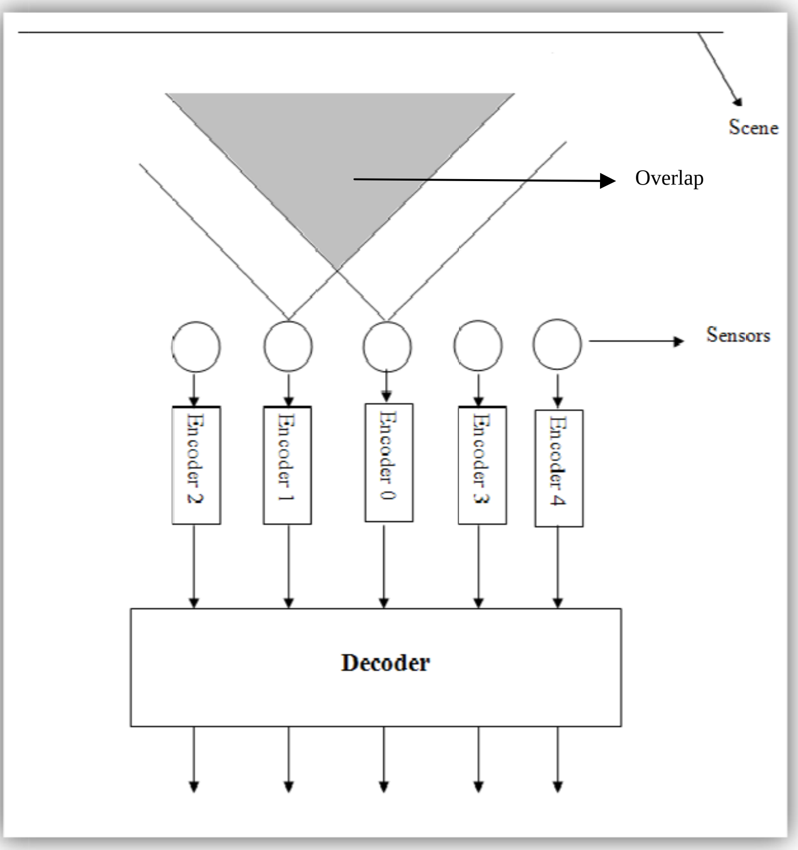


Figure 4 - Block diagram of distributed sensors with overlapping fields of view

Chapter 3

Specifications of the encoders and the decoder

3.1 The Encoders

In the previous section we showed an overview of how we are going to implement the distributed coding system. In this section we will go through every section of the algorithm in detail and describe all the procedures. First we start by showing some figures on how our two encoders will look like:

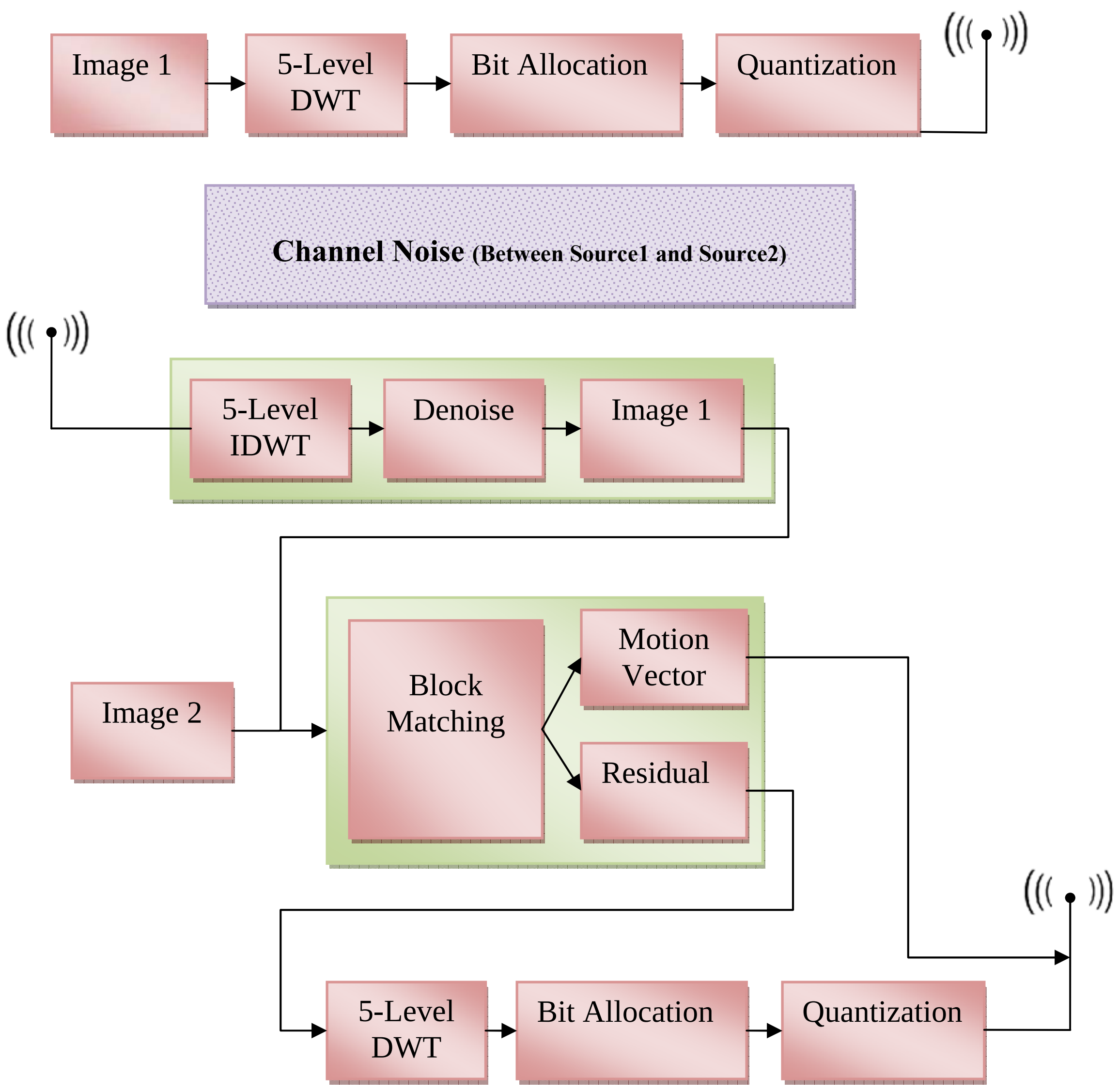


Figure 5 – The Block diagram of the two encoders

3.1.1 Discrete Wavelet Transform

Before describing each block in detail, we need to explain why we have chosen Discrete Wavelet Transform for image compression [11], [12]. The choice of transform can be made in many ways with the objective of optimizing some property such as signal-to-noise ratio of the image, edge enhancement, image coding or compression. Wavelet transform is of interest for the analysis of non-stationary signals because it provides an alternative to classical short-time Fourier transform or Gabor transform. For periodic signals Fourier analysis is ideal. However with wavelet transforms we are not restricted to only the periodic functions but any function.

Unlike the Fourier transform which maps a 1-D signal to a 1-D transform domain, wavelet transform maps a 1-D signal to a 2-D transform domain. A wavelet transform thus has a highly redundant number of basis functions. In many cases of signal processing, one can choose the right signal or a theoretical model as the mother wavelet. The advantage of doing is that only a few wavelet transform coefficients are then required to represent the signal i.e. signal compression.

The proof of the maths behind wavelet transform is not the objective of this project and only properties of wavelet transforms in image processing are stated. Figure 6 shows a block diagram of a single-level wavelet transform decomposition of two dimensional signals and Figure 7 shows its inverse wavelet transform (reconstruction of the signal).

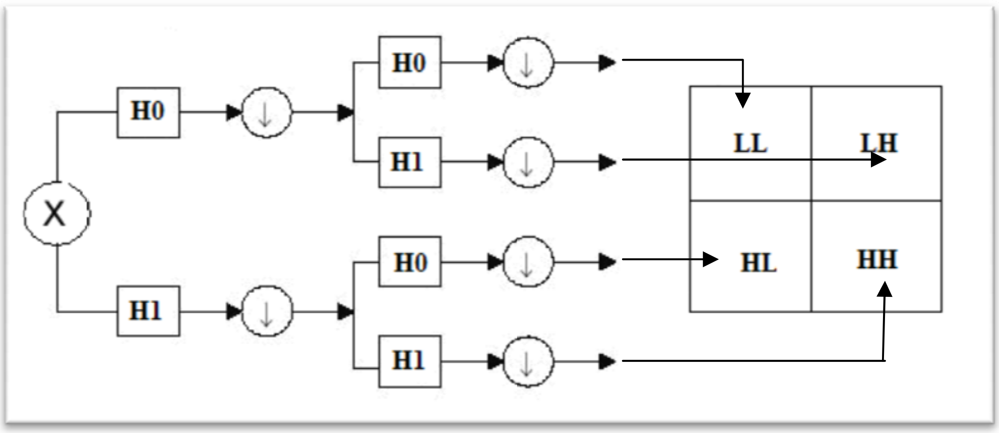


Figure 6 – 2-D Forward Wavelet Transform, also known as the Analysis section

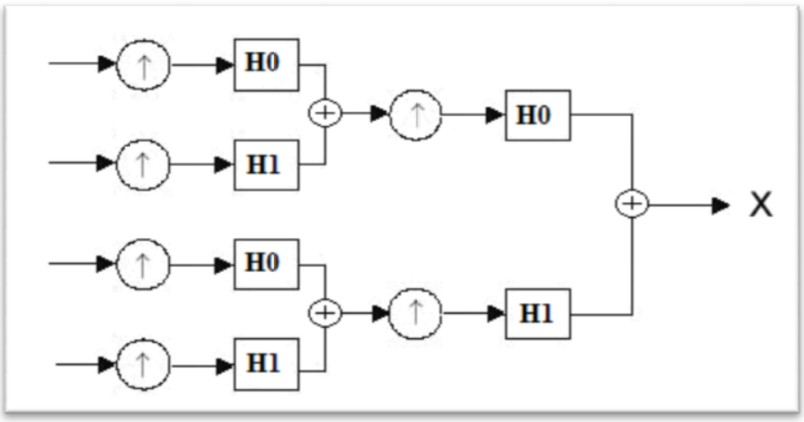


Figure 7 – 2-D Inverse Wavelet Transform, also known as the Synthesis section

Here H_0 represents a low-pass filter and H_1 represents a high-pass filter. This will decompose a two dimensional signal, for example an image, into four sub-images of equal size. The top-hand left image is the low resolution of the original image and the rest of the images being the vertical, horizontal and diagonal components of the original image. Usually the low resolution part has over 90% of the energy of the image. Now we can extend this decomposition on the low resolution part and get another 4 sub-images. This is called the second-level decomposition. Examples of single-level and second-level wavelet decompositions on the original image in figure 8 are shown in figures 9 and 10 respectively. We can even go further and do 5-level decomposition. Figure 11 shows the 5th level wavelet decomposition of the original image.

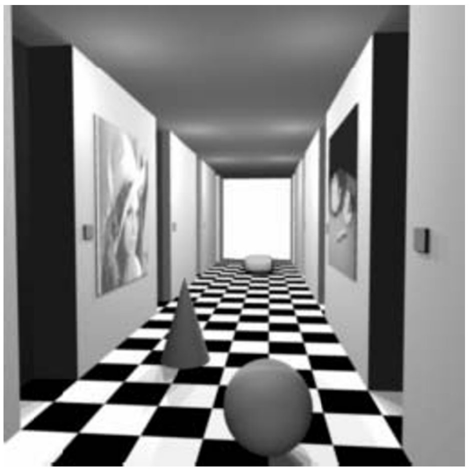


Figure 8 – The Original Image "Room" of size 512x512

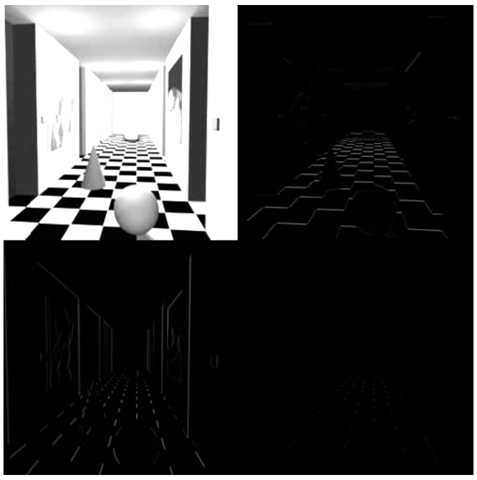


Figure 9 - First Level Wavelet Transform Decomposition

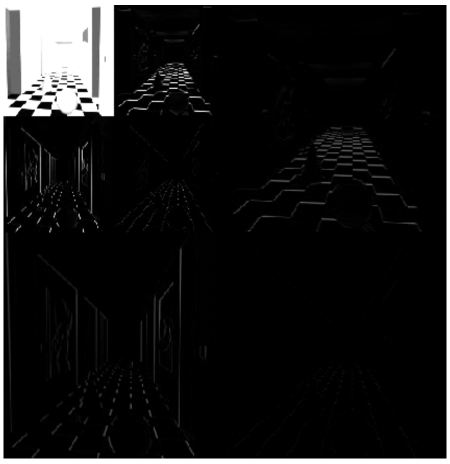


Figure 10 - Second Level Wavelet Transform Decomposition

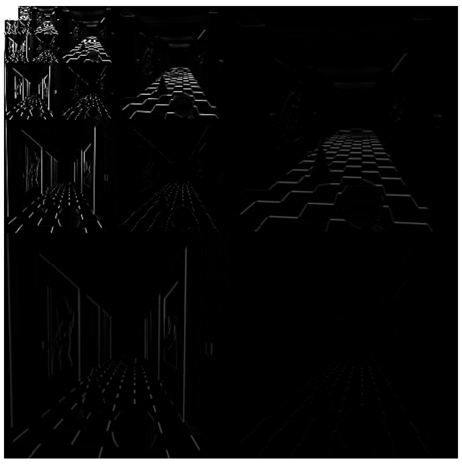


Figure 11 – 5th Level Wavelet Transform Decomposition

We can run a simple test to show how easy it is to do compression on images using wavelet transform. First we do a two-level wavelet transform decomposition of the image shown in Figure 8. Then we disregard all the vertical, horizontal and diagonal parts of the transform i.e. we ignore 94% of the wavelet coefficients assuming that the image is 512x512. Then we take the inverse wavelet transform. The resulting image is shown next to the original image in Figure 12. As can be seen from the figure, the compressed image is not that different from the original image with 94% of the coefficients being disregarded. The reader should bear in mind that we are not using any other kind of compression scheme such as Quantizing, Run-length coding, Entropy coding etc. This shows why wavelet transforms are so powerful in terms of energy compaction.

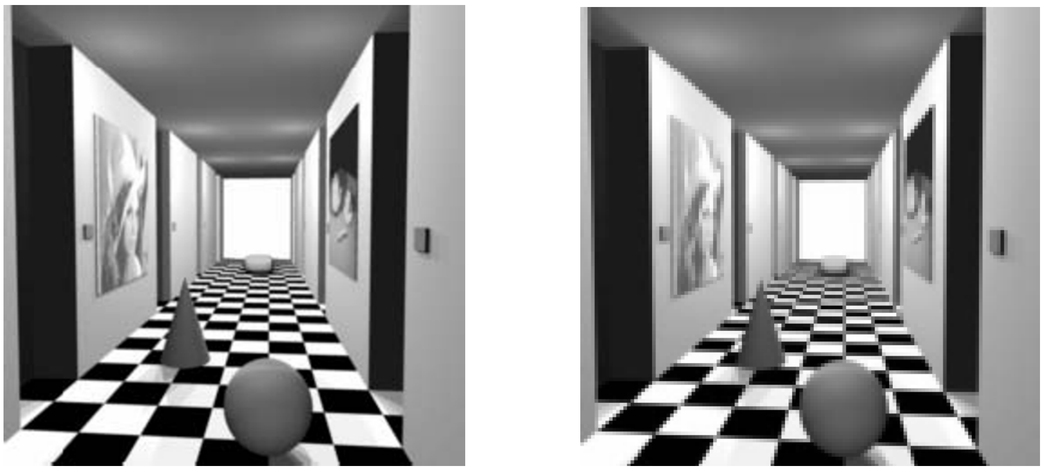


Figure 12 – The Original image of "Room" of size 512x512 and its compressed version. 94% of the wavelet coefficients are neglected

3.1.2 Comparison of Images

One way of comparing the original image with its compressed version is to find the Peak signal-to-noise ratio (PSNR). PSNR is the most common used measurement tool for comparing images. It is calculated as follows:

Distortion[1] = $(Original\ Image\ .-\ Compressed\ Image)^2$

Where ".$-$" is the sign for element by element subtraction

Mean Squared Error (MSE) = $\frac{Sum\ of\ all\ the\ elements\ of\ the\ Distortion\ natrix}{Total\ number\ of\ pixels}$

PSNR = $20 \times \log\ (255/sqrt(MSE))$ dB

Where 255 is the peak value of any 8-bit image.

For example, the PSNR of the compressed image in figure 12 is 22.53dB

3.1.3 Wavelet Filters

The choice of filters used is quite important in wavelet processing. For example, for the wavelet decompositions shown above, we used Haar filters. Haar filters or “sum and difference” filters are the most elementary but at the same time the most popular filters because of their simplicity. The Haar filters are defined as follows (in z-domain):

$$H_0(z) = \frac{1+z^{-1}}{\sqrt{2}} \qquad H_1(z) = \frac{1-z^{-1}}{\sqrt{2}} \qquad G_0(z) = \frac{1+z}{\sqrt{2}} \qquad G_1(z) = \frac{1-z}{\sqrt{2}}$$

Some other famous filters used in wavelet theory are Daubechies, B-Spline, Symlets, and Coiflets. The filters are chosen depending on where we want to use them. For image compression and denoising, Haar and Daubechies filters are the most popular.

[1] Distortion could also be calculated as $|Image1\ .-\ Image2|$ where |argument| is the absolute value of the argument

3.1.4 Quantization and Inverse Quantization

After finding the N-th level wavelet transform of the image, we need to quantize the wavelet coefficients due to the limited channel bandwidth. In this project we will use a simple uniform quantizer. As we already know, quantization is a lossy process. By transforming an image to its wavelet domain we will have different classes of wavelet coefficients. The quantization process is done on each class of the wavelet coefficients, that is, from the first until the N-th level vertical, horizontal and diagonal components. The approximation subband will also be quantized. The quantization is done as follows:

1) First we find the maximum and minimum of a given class (say the first level vertical component)
2) Then we add the absolute value of the minimum to all the coefficients of that class. This is done so that we won't worry about negative coefficients in terms of assigning bits.
3) After that, we find the step size at the given number of bits or we find the number of bits at a given step size. We decided to assign the bit number instead of the step size as it is easier to compare results at an exact bit number. The formula for uniform quantization is $Max = 2^{\gamma} \times \Delta$ where Q =2^{γ} is the number of quantization levels, γ is the number of bits and Δ is the step size. By knowing gamma we can find the step size from the formula above.
4) Now we divide all the coefficients of the considered class by the step size.
5) Finally we round up all the quantized coefficients. This step is lossy and we get the quantization noise from here.

At the decoder side we need to inverse quantize or "dequantize" our quantized data. This is done by reversing whatever that has been in the quantizer. One thing that is quite important to mention here is that we need to store the values of minima and maxima in the quantization process. These data will be sent to decoder side with the quantized data for inverse quantization. The number of bits assigned to the maxima and minima values is in the order of 0.001-0.002bpp (for a 512x512 image with 5-level wavelet decomposition). Therefore for sending the maxima and minima values to the decoder side, very small number of bits is needed.

Now the question that arises here is that how do we assign the number of bits to each class of the wavelet coefficients?

3.1.5 Bit Allocation

We need to decide on how to assign the number of bits to different blocks (classes) of the transformed image at a given bit-rate. The idea of bit allocation is to optimally assign the number of required bits for a given class of wavelet coefficients which will result in the least amount of distortion. One way to optimally assign bits to different classes of wavelet coefficients is defined as follows [13], [14]:

1) Initially all the classes are allocated with a predefined maximum number of bits.
2) For each class we reduce one bit from it and we find the distortion due to reduction of that 1 bit. This is done by quantizing and inverse quantizing every single class and finding the MSE with the original class data.
3) The number of bits of the class that has the least distortion will be reduced by 1 bit.
4) The total rate R is calculated as total sum of p * b for each class where p is the probability of that class and b is the number of allocated bits to that class. The probability here means the total number of pixels of the class over the total number of pixels of the image which in our case is 512x512.
5) Finally we compare the calculated R with the required bit-rate. If R is higher we go back to step2.

The bit allocation process defined above is quite complex in terms of computation especially if the target bit-rates are very small. When the target bit-rates are small we can use an alternative approach. In this approach, which is also mentioned in [14], we work backwards and we assign 0 bits to all classes. The procedure is as follows:

1) First we assign 0 bits to all classes
2) Then we add one bit to each class and the deduction in distortion due to that 1 bit is calculated.
3) The number of bits of the class that has the highest deduction in distortion will be added by one.

4) We calculate the total rate R as before. If R is smaller we go back to step2.

As can be seen, the second approach will be much faster if we are dealing with small bit-rates. Now, quantization is very easy after finding the allocated number of bits for each class of wavelet coefficients. For example for 5-level wavelet decomposition our gamma matrix will look like: [H1 V1 D1 H2 V2 D2 H3 V3 D3 H4 V4 D4 H5 V5 D5 A] where H, V, D and A stand for Horizontal, Vertical, Diagonal and Approximation (low resolution part) respectively.

Now having talked about the bit allocation problem we can show some examples of images compressed with our codec system. Figure 13 shows the original "Room" image. Figures 14, 15 and 16 show the image at the compression rates of 1bpp, 0.5bpp and 0.1bpp i.e. 88%, 94% and 99% compression respectively (the original image is 8bpp). We used 5-level wavelet decomposition with Haar filters for compressing the original image. The corresponding allocated bit-rates for all the classes are as follows:

At 1bpp:	[0 1 0 3 4 0 5 6 4 7 7 7 8 8 7 8]
At 0.5bpp:	[0 0 0 2 2 0 4 5 3 7 6 6 7 7 6 8]
At 0.1bpp:	[0 0 0 0 0 0 0 1 3 3 3 1 4 4 4 5]

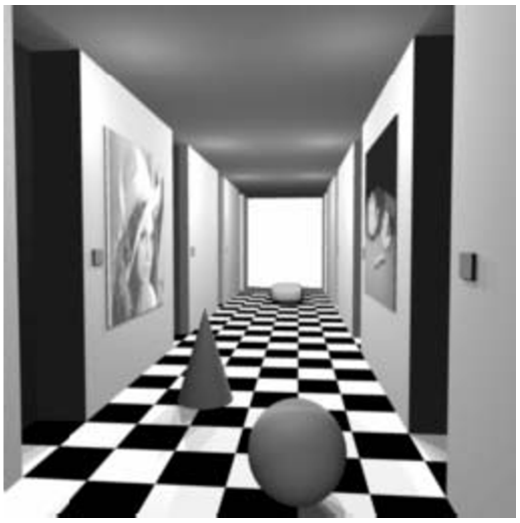


Figure 13 – The original "Room" picture with size 512x512 with 8bpp

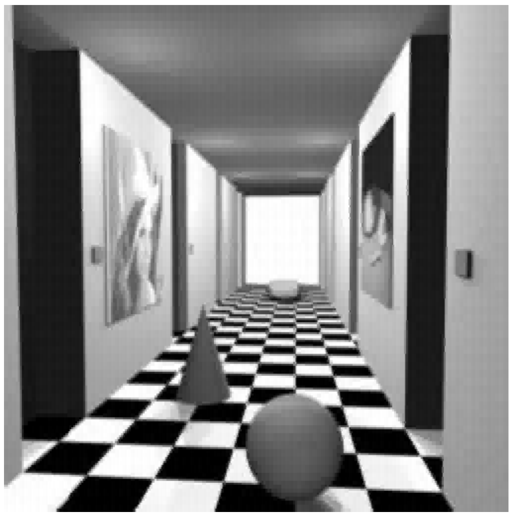


Figure 14 – The compressed picture at the rate 1bpp, PSNR = 27.98dB

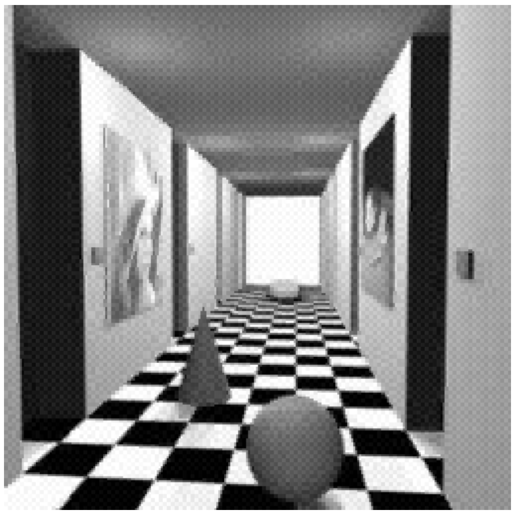


Figure 15 - The compressed picture at the rate 0.5bpp, PSNR = 25.42dB

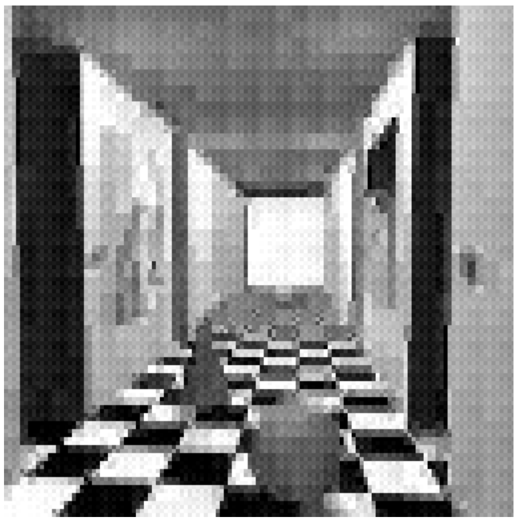


Figure 16 - The compressed picture at the rate 0.1bpp, PSNR = 18.11dB

What we have talked about so far is quite similar to what the JPEG2000 codec system does. The reader has to bear in mind that JPEG2000 has so many other features that are outside the scope of this project. One important thing to mention here is that the encoders need also to send the bit numbers for the all classes. The number of bits assigned in the gamma matrix is in the order of 0.0001bpp (for a 512x512 image with 5-level wavelet decomposition). Therefore the gamma matrix needs very few bits for it to be sent to the decoder.

3.1.6 Encoder1 - The Reference Image

After defining our codec system now we need to explain how we are going to compress our two stereo images as efficiently as possible in a distributed fashion. So let's again show the block diagram of our system:

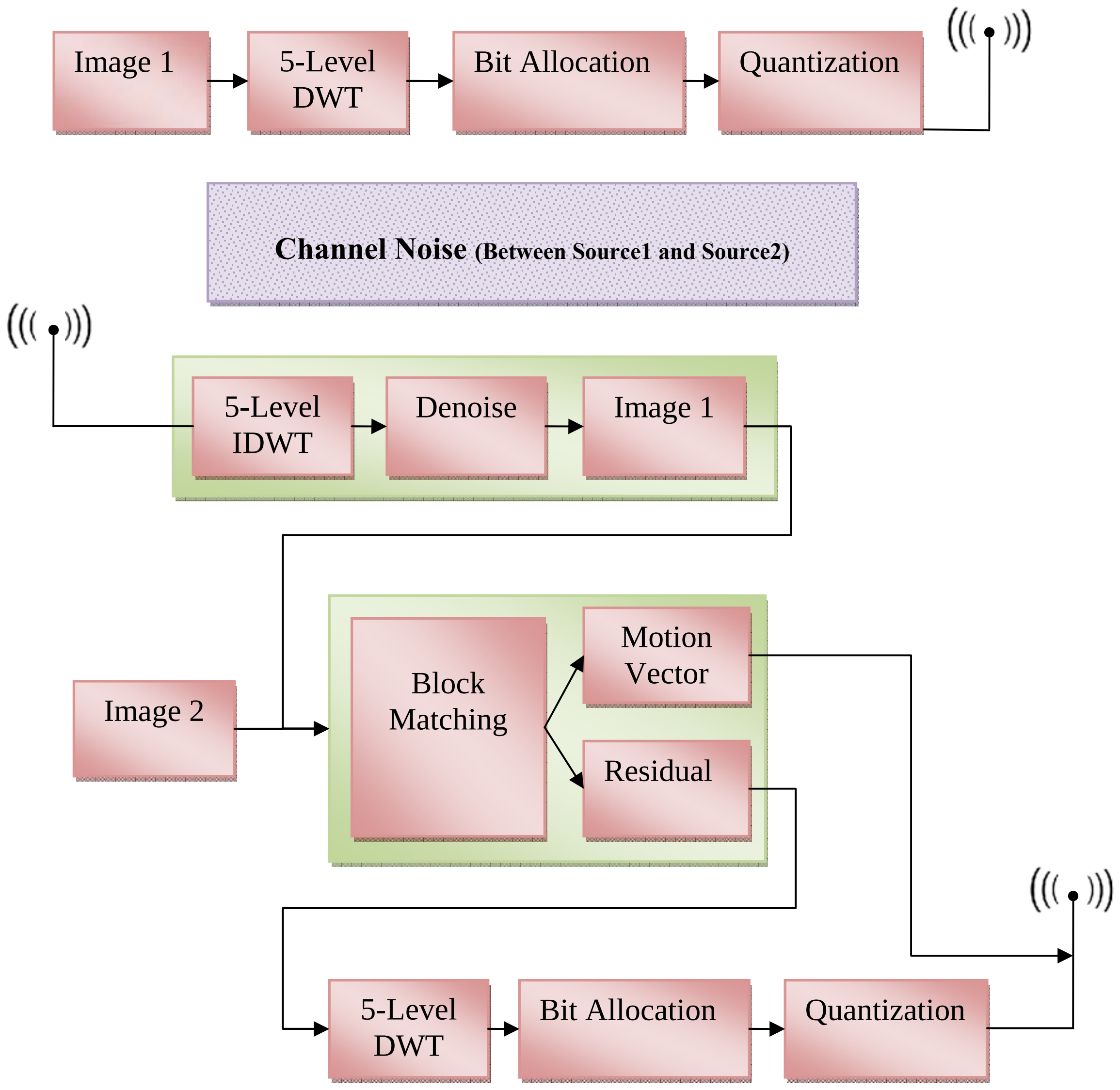


Figure 17 – The Block Diagram of the Encoder1 and Encoder2

As mentioned before the reference encoder needs to send the complete coded version of the captured Image1 to the decoder regardless of what other cameras are viewing. Here, we assume that the raw image taken from the cameras is already digitized. As shown in the figure we first take the wavelet transform of the captured image then we allocate bits for each class of wavelet coefficients. After the completion of bit allocation process, the wavelet coefficients are quantized according to the gamma matrix. Finally the coded image is sent directly to the receiver.

3.1.7 Encoder 2 - The Adjacent Camera

The second encoder overhears a noisy and coded version of Image1. Therefore, as stated before, we need to take the inverse quantization and the inverse wavelet transform of the incoming image and then denoise it. After denoising, we will use motion compensation method to find the motion vector of image2 with respect to image1. Then we find the residual image by subtracting the original image2 from its estimated version. Finally we send the motion vector with the coded residual image.

3.1.8 Denoising

We will use an adaptive wavelet threshold estimation method, called the NormalShrink denoising algorithm [15], to denoise our noisy Image1. It is assumed that the probability density function of the channel noise is Gaussian distributed. Since we are taking 5-level wavelet decomposition, there will be 16 classes of wavelet coefficients ([H1 V1 D1 H2 V2 D2 H3 V3 D3 H4 V4 D4 H5 V5 D5 A]). The denoising algorithm is applied to the first 15 classes and the approximation subband is left intact.

The proposed formula in [15] for the threshold is:

$$T = \frac{\beta \hat{\sigma}^2}{\hat{\sigma}_y}$$

Where beta is the scaling parameter and is defined as: $\beta = \sqrt{\log\left(\frac{L_k}{J}\right)}$. Here, L_k is the length of the subband (class) and J is the number of decompositions, which in our case is 5. $\hat{\sigma}_y$ is the standard deviation of the subband being considered and $\hat{\sigma}^2$ is the variance of the noise which is estimated form the subband D1. The formula that is used in [15] for estimating the variance of the noise is: $\hat{\sigma}^2 = \left(\frac{median(|Y_{ij}|)}{0.6745}\right)^2$. Here, 0.6745 is a normalisation factor so that the variance of the noise will correspond to the variance of Gaussian distribution.

Once the threshold is found we can apply either soft thresholding or hard thresholding for denoising. The formulas for soft and hard thresholding for a given coefficient and threshold are as follows:

Soft Thresholding => $Coeff = sign(Coeff) \times \max(0, |Coeff| - Threshold)$

Hard Thresholding => $Coeff = \begin{cases} Coeff & if\ Coeff \geq Threshold \\ 0 & Otherwise \end{cases}$

Generally soft thresholding gives better results in wavelet denoising. Soft thresholding moves the coefficients towards zero and results in a smooth image which could help us in motion compensation (next section).

A problem that we face here is that for small bit-rates the subband D1 is always zero so we cannot estimate the variance of the noise. One way to overcome this problem could be to estimate the noise from D2, D3, D4 or D5 subbands but after some tests we realized that the estimated variance will be quite large and finding a normalizing factor which would work for any given image would not be straight forward and may not be accurate. Therefore we assumed that the second encoder knows about the variance of the channel noise beforehand. This could be done by sending a small known signal from encoder 1 to encoder 2. Encoder2 can then compare the received message with the original (known) message and estimate the channel noise with a good accuracy.

Now we will show some examples on how our denoising algorithm performs under different noise variances. Figure 18 and 19 show the noisy and denoised “Room” image at $\sigma = 20$ where σ is the standard variation of the channel noise. The noise is added to the image by the “randn” function in MATLAB. The mean of “randn” function is 0 and its variance is 1. By using the expression $Var(aX) = a^2 Var(X)$ we can generate Gaussian distributed noise with any variance that we wish. For example if we want a noise variance of 100 we just multiply our “randn” function by the square root of 100 i.e. 10.

Figure 20 and 21 show the noisy and denoised image at $\sigma = 30$. Also Figure 22 and 23 show the noisy and denoised image at $\sigma = 40$. From the figures and the PSNR values we can see that the wavelet thresholding denoising algorithm performs quite well.

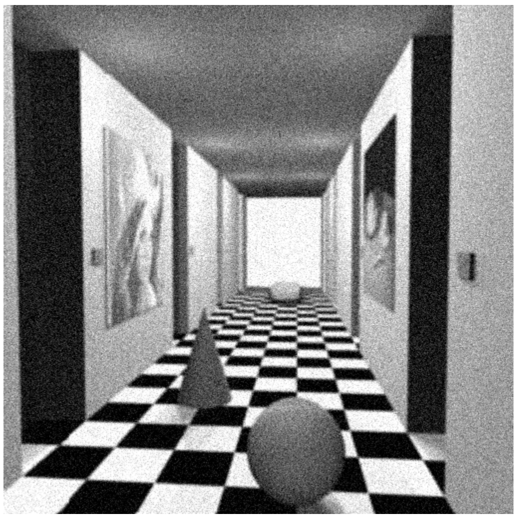


Figure 18 – The noisy image, σ =20, PSNR = 22.10dB

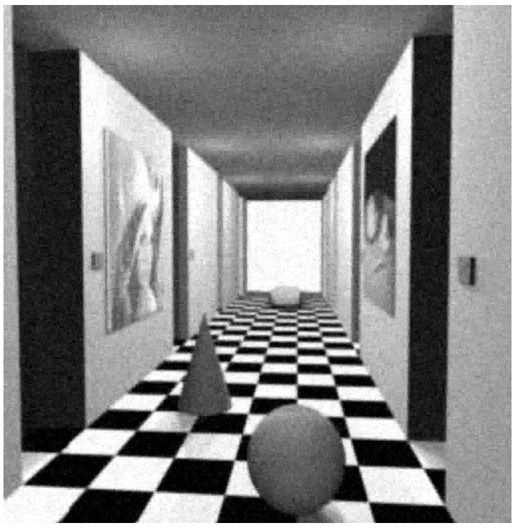


Figure 19 – The denoise image using wavelet, thresholding, PSNR = 27.69dB

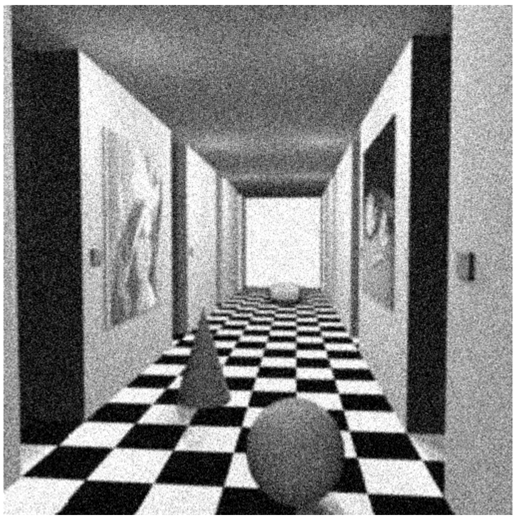


Figure 20 – The noisy image, σ =30, PSNR = 18.58dB

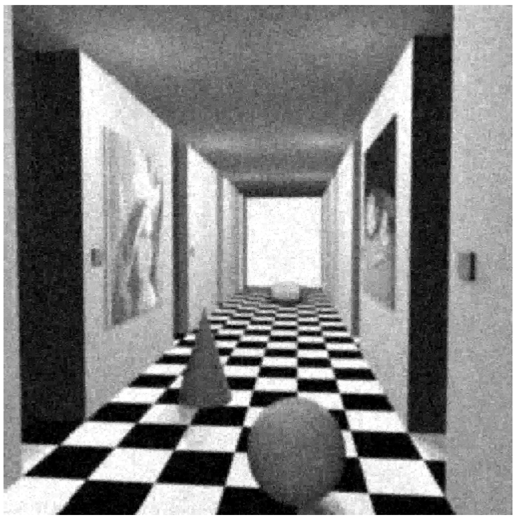


Figure 21 - The denoise image using wavelet, thresholding, PSNR = 25.07dB

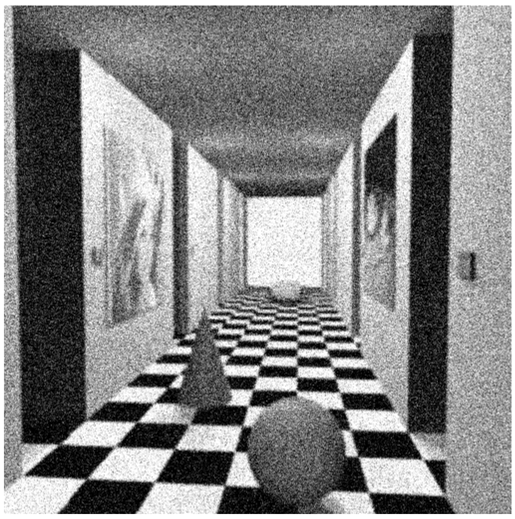


Figure 22 – The noisy image, σ =40, PSNR = 16.09dB

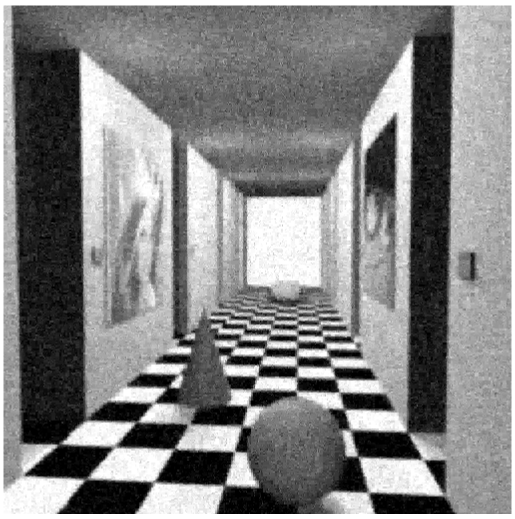


Figure 23 - The denoise image using wavelet, thresholding, PSNR = 23.25dB

One thing that is quite important to mention here is that for very noisy images, wavelet transform itself (without any thresholding) could be used for denoising. This is done by eliminating the diagonal or even horizontal and vertical subband of the wavelet transform. The reason is that by eliminating the diagonal subbands, we are eliminating high frequency noise. This is very evident when we are working at very low bit-rates. At low-bit rates, the bit allocation process will only allocate bits to subbands that will result in the least distortion, therefore most of the high frequency terms are set to 0. This will result in a better image quality. Figures 25 and 26 show what happens to our noisy image at 1bpp bit-rate and Figures 27 and 28 show what happens to our noisy image at 0.2bpp.

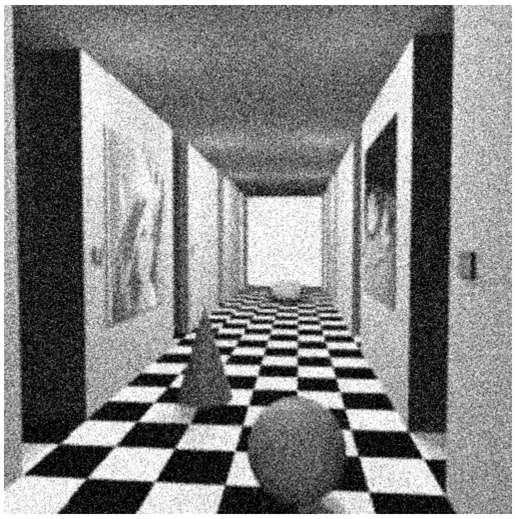


Figure 24 – The original noisy image, σ =40, PSNR = 16.09dB

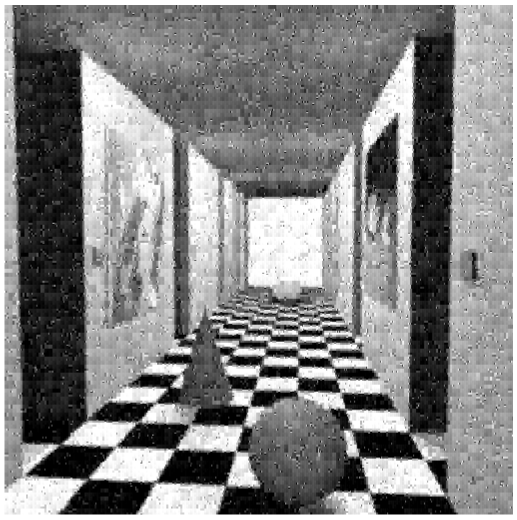


Figure 25 - The noisy image at 1bpp, PSNR = 19.20dB

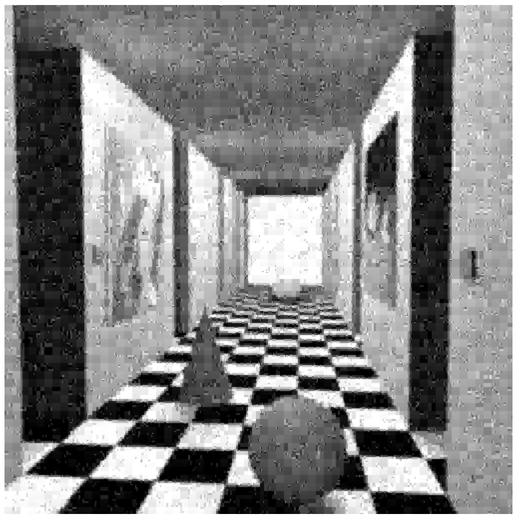


Figure 26 – The denoised image at 1bpp, PSNR = 21.55dB

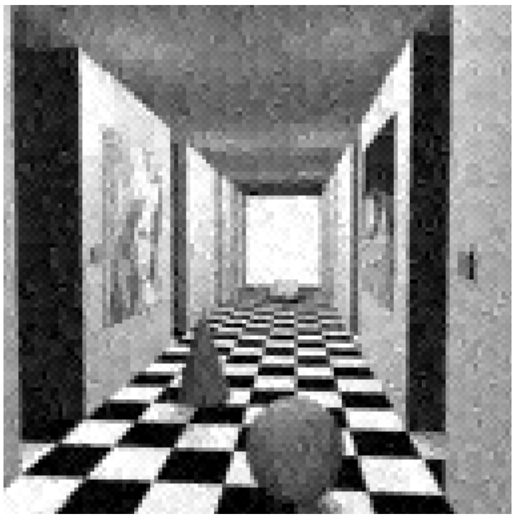


Figure 27 - The noisy image at 0.2bpp, PSNR = 21.00dB

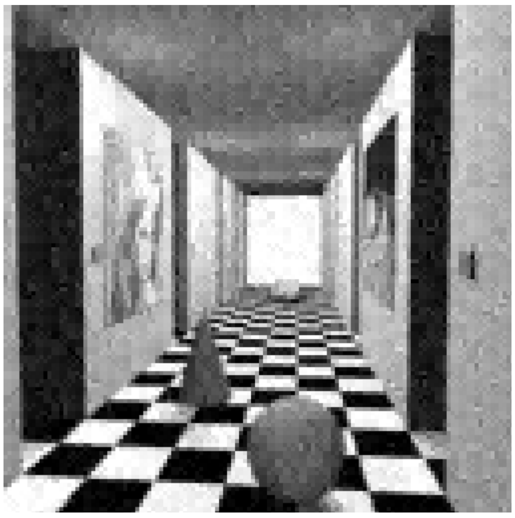


Figure 28 - The denoised image at 1bpp, PSNR = 21.28dB

We can see from the figures that our noisy image has better quality at 1bpp and 0.2bpp than the case with 8bpp (no quantization). Also we can see from figures 27 and 28 that at very low bit rates (0.2bpp here) the denoised image is really no different from the noisy image. This is because most of the denoising is done on the diagonal subbands of the wavelet transform and at low bit-rates these are set to zero. We need to mention here that we used Haar filters for wavelet denoising.

3.1.9 Block Matching and Motion Vector [16]

After decoding and denoising the received signal from encoder1 we will use motion compensation to find an estimated version of captured image2 from image1. In this method we divide up the captured image2 into non-overlapping blocks of KxK and we estimate the motion compensation vector, one for the horizontal shift and one for the vertical shift. After that we compare a block in image2 with the same block in image1 and its surrounding blocks. By surrounding blocks we mean a rectangular search box around the block in Image1. The block that has the least distortion with regards to the considered block in image2 is chosen. The distortion here is calculated by taking the absolute difference of the two blocks.

The positions of these blocks are stored in a matrix called the motion vector. By using motion compensation we only need to send the two motion vectors and the disparity between the original captured image2 and its estimated version. The disparity will be compressed before being sent to the decoder.

The choice of block size and the search area (surrounding blocks) is dependent on the images we are working on. If the images are too complex, that is if they are highly detailed, then we need to use small block sizes. The choice of block size is a trade-off between accuracy and bandwidth. The smaller the block size the better the estimation and the bigger motion vector. Usually blocks of 16x16 and 8x8 are used.

The rectangular search area is dependent on how far apart the cameras are. If they are very close to each other then we can use a small search area. If they are quite far from each other then we need to either expand our search area or shift the search area according to the distance of the cameras. In this project we assume that the cameras are quite close to each other. The choice of search box size is a trade-off between accuracy and speed. The larger the box size the more computationally complex it will get. With the test images we used anything between 5x5 and 9x9 search boxes would give good results both in terms of accuracy and speed.

The intensities of the pixels of the residual image are usually close to zero especially if the correlation between the two images is high. The range of the residual image is usually between -255 to 255 but the mean is close to zero. This could be a problem at low bit rates since, generally the number of pixels that are above 128 and below -128 is very small and as we are using a uniform quantizer we can get poor results. One possible way to deal with this problem is to keep the residual image range from -128 to +128. Therefore any pixel that is higher than 128 changes to 128 and any pixel that is lower than -128 changes to -128. At low bit rates, this will improve the performance of the distributed coding and joint coding significantly.

Now we can show an example of how the estimated version of image2, the residual and the motion vector will look like. Figure 29 and 30 show the original “Room Left” and “Room Right” stereo pictures respectively. Figure 31 shows how the estimated version of “Room Right” will look like from “Room Left” with block size of 16x16 and search area of 9x9. Figures 32 and 33 show the corresponding residual image and the motion vectors. Here we are assuming that the received Image1 in the encoder2 is neither noisy nor compressed i.e. Joint Coding with no quantization.

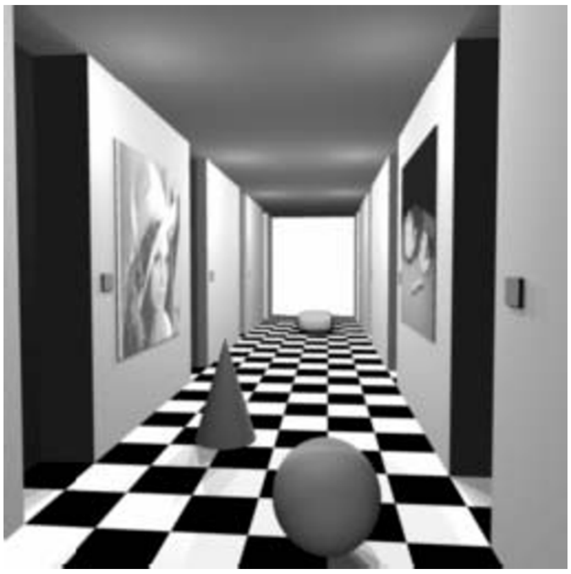


Figure 29 – Original "Room Left"

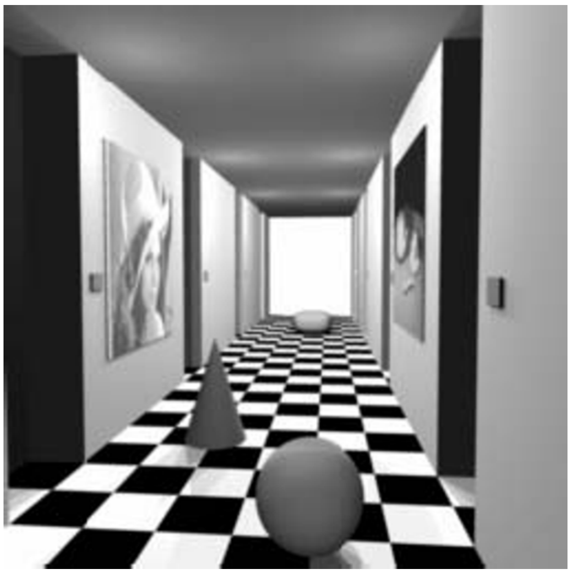


Figure 30 - Original "Room Right"

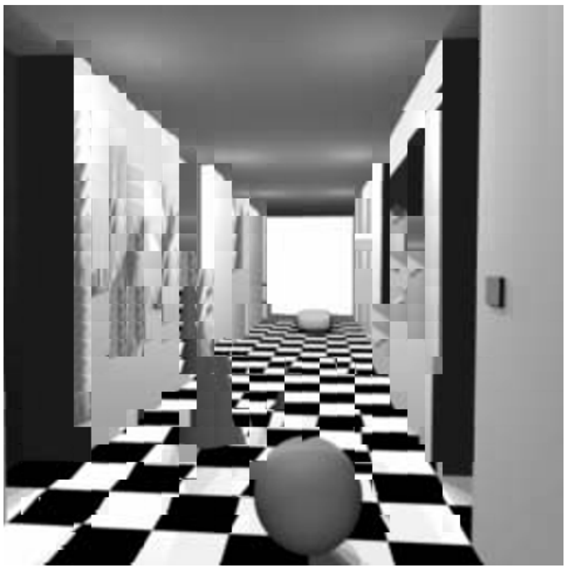


Figure 31 – Estimated image of "Room Right"

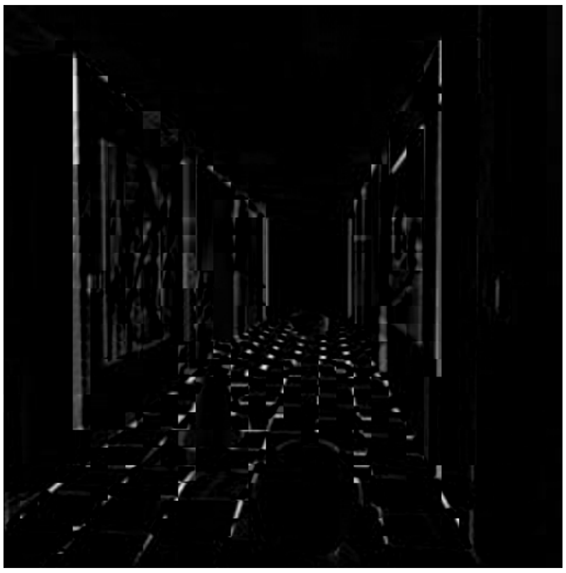


Figure 32 – The residual Image

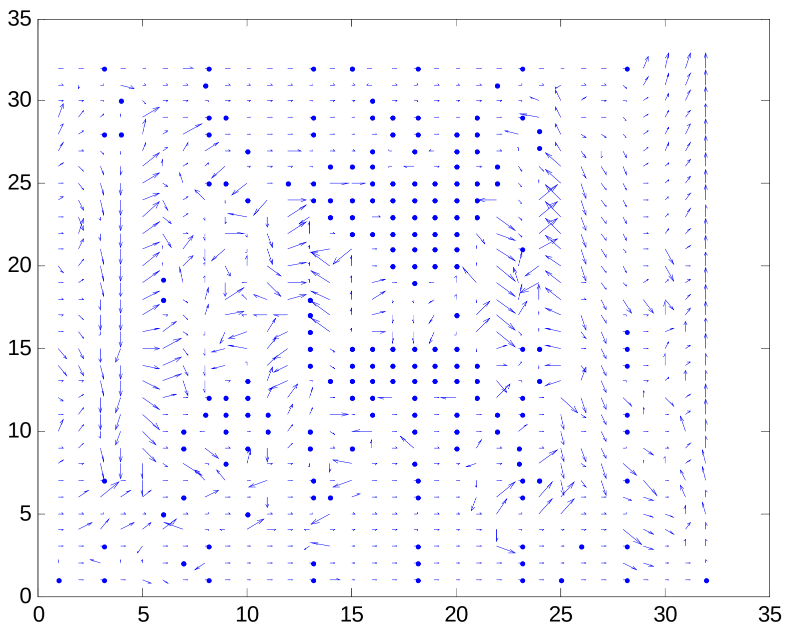


Figure 33 – The motion vector

In the decoder, we have the complete version of “Room Left” image. From this image and the motion vector we can generate the estimated version of Image2. By adding the residual image to the estimated image we can reconstruct our Image2. The choice of block size is very important here. Figure 34 and 35 show the estimated Image 2 for block sizes of 8x8 and 32x32 respectively. As can be seen from the figures with block size of 8x8 the estimated version is very close to the original image. The difference will be even more evident when we deal with more complex images (high detailed images).

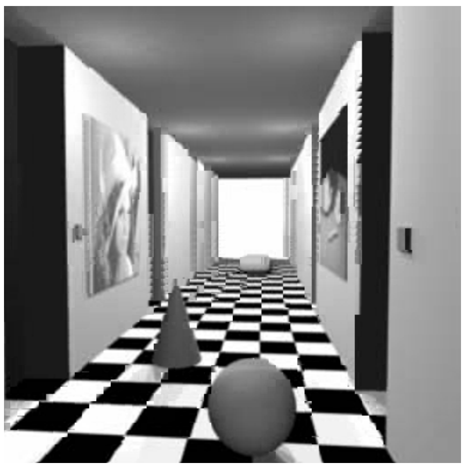


Figure 34 - Estimated image of “Room Right”, with block size of 8x8

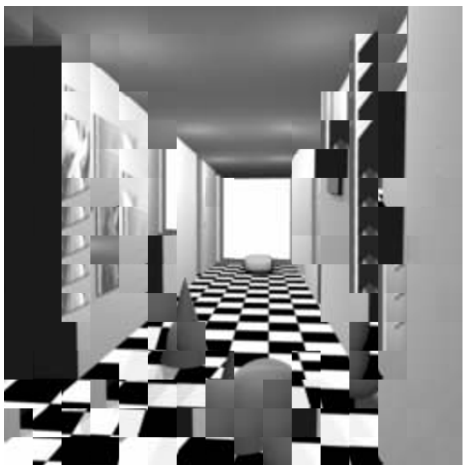


Figure 35 - Estimated image of “Room Right”, with block size of 32x32

Figure 36 and 37 show the original stereo image pair of Pentagon. Figure 38 and 39 show the estimated version of "Pentagon Right" with block size of 8x8 and 16x16 respectively. Although with block size of 8x8 more bits is needed to send the motion vector[2] to the decoder but generally block size of 8x8 gives better results in rate-distortion graphs.

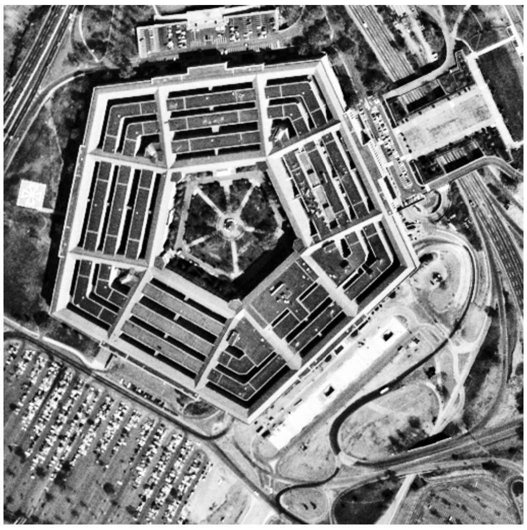


Figure 36 – Original "Pentagon Left"

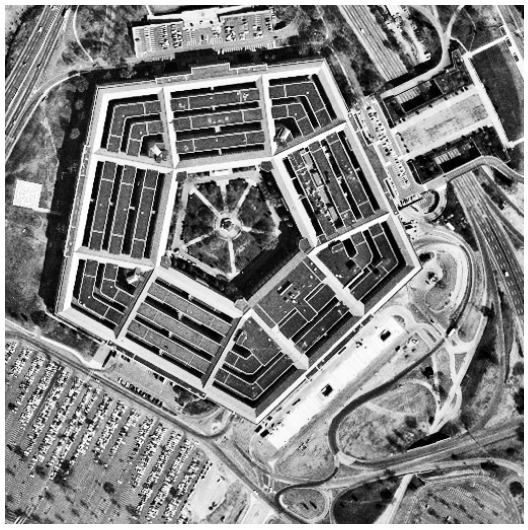


Figure 37 - Original "Pentagon Right"

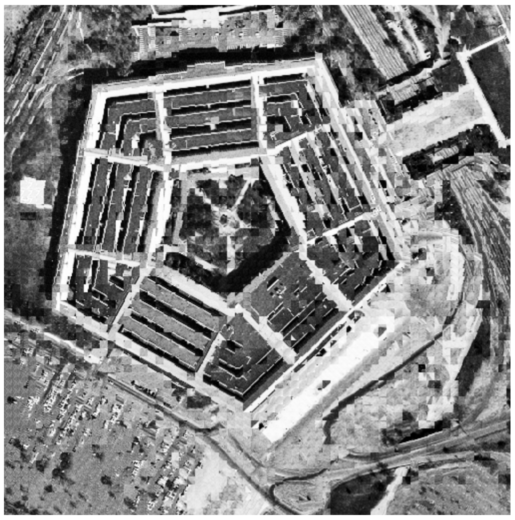


Figure 38 – Estimated image of "Pentagon Right" with block size of 8x8

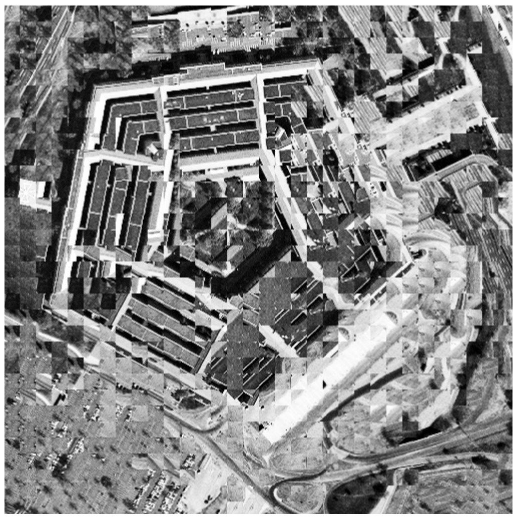


Figure 39 – Estimated image of "Pentagon Right" with block size of 16x16

[2] The size of horizontal and vertical motion vectors will be 64x64 each. The stereo images are 512x512 pixels.

3.2 The Decoder

At the decoder we receive the complete version of Image1 and the motion vector with the residual image of Image2. First of all, the decoder needs to do inverse quantization and inverse wavelet transform on both Image1 and the residual image. Then it will generate an estimated version of Image 2 from the motion vector and Image1. Finally the decoded residual image is added to the estimated image for reconstruction. Figure 40 shows a block diagram of the Decoder:

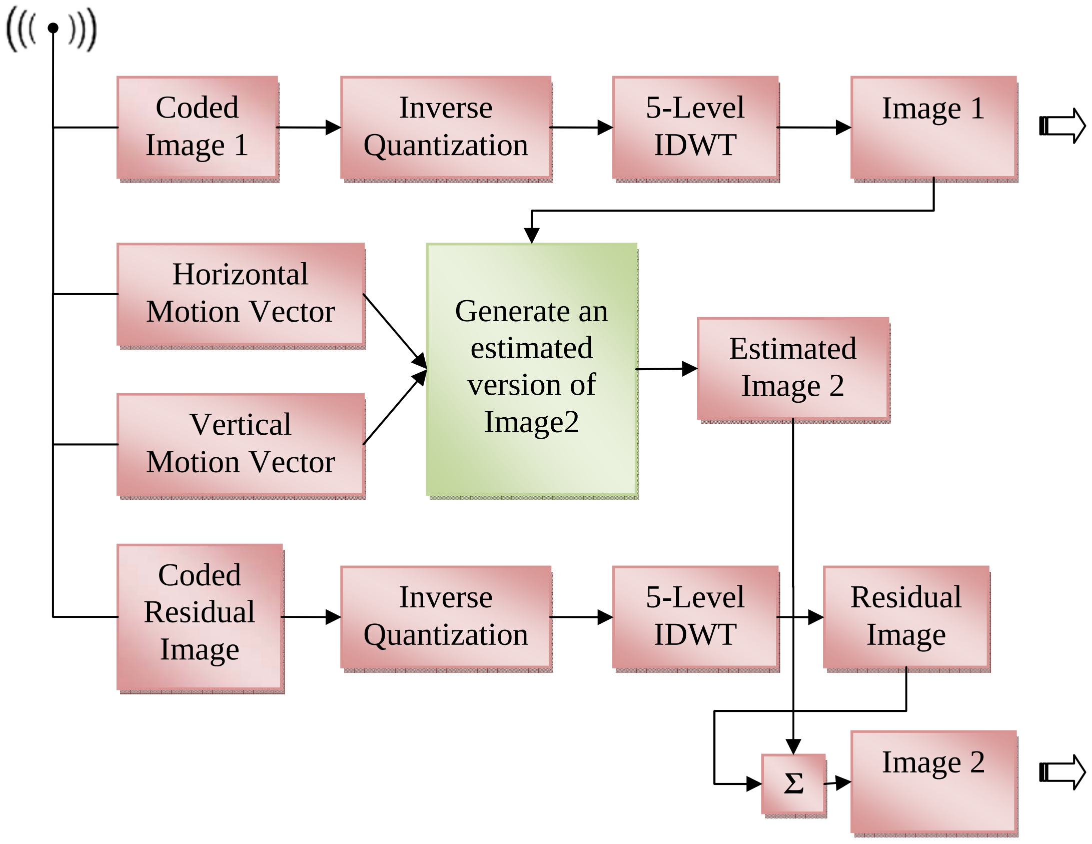


Figure 40 – Block diagram of the Decoder

Chapter 4

Results and Analysis

We used MATLAB for programming and all the codes are available in the Appendix. We will test the proposed system with some stereo image pairs. As mentioned before our stereo images are 512x512 pixels and greyscale.

The aim here is to see how close we can get to the optimum result with our distributed coding system with limited and corrupted communication between the encoders. The optimum case (the upper boundary) for our system is when there is no channel noise between encoder 1 and encoder 2. This is basically just joint source coding where all the cameras have all the information available from their neighbouring cameras. We can also have a lower boundary. The lower boundary could be when we independently compress our two images without exploiting the correlation between the two images.

We chose 5 stereo image pairs for testing. We added different noise standard deviations of σ = 10, 20 and 30. $\sigma = 0$ means that there is no channel noise, so this will be used for joint source coding. For Image1 bit-rates of 4bpp, 2bpp and 1bpp were chosen for testing with each stereo pair. Also the working bit-rate range for Image2 was chosen to be within 0 to 0.6bpp.

4.1 Stereo Pair 1 - Room

Figures 41 and 42 show the original stereo pictures of “Room Left” and “Room Right” respectively. Figures 43, 44 and 45 show the rate-distortion graphs for Image2 when Image1 is sent to the decoder at the rates 4bpp, 2bpp and 1bpp respectively. In these tests the standard deviation of noise is 10. The corresponding PSNRs of Image1 are 43.52dB at 4bpp, 33.93dB at 2bpp and 27.98dB at 1bpp.

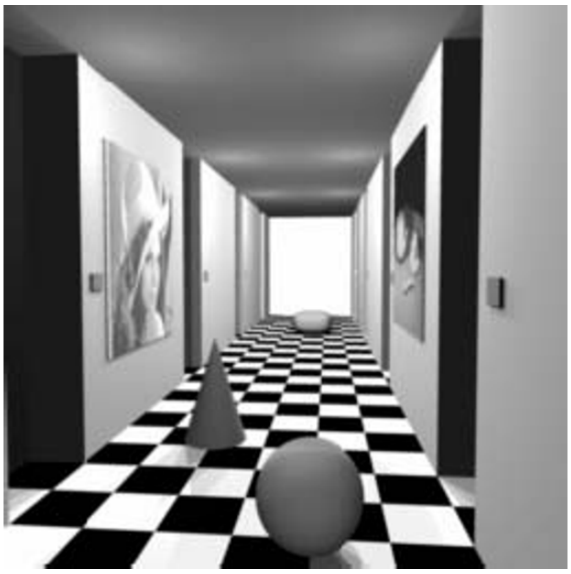


Figure 41 – Original “Room Left” image

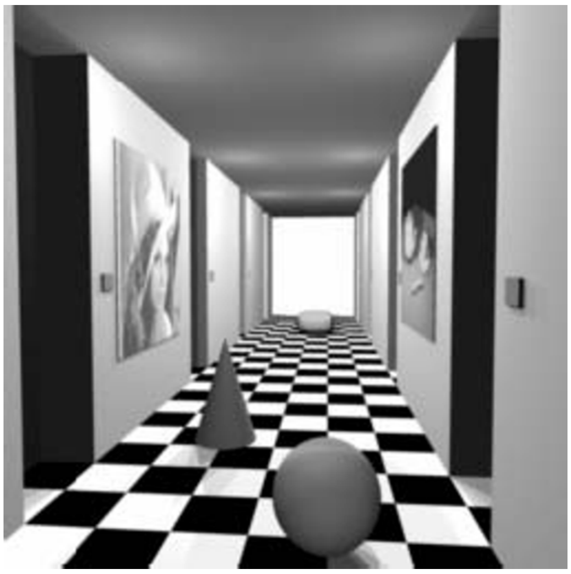


Figure 42 – Original “Room Right” image

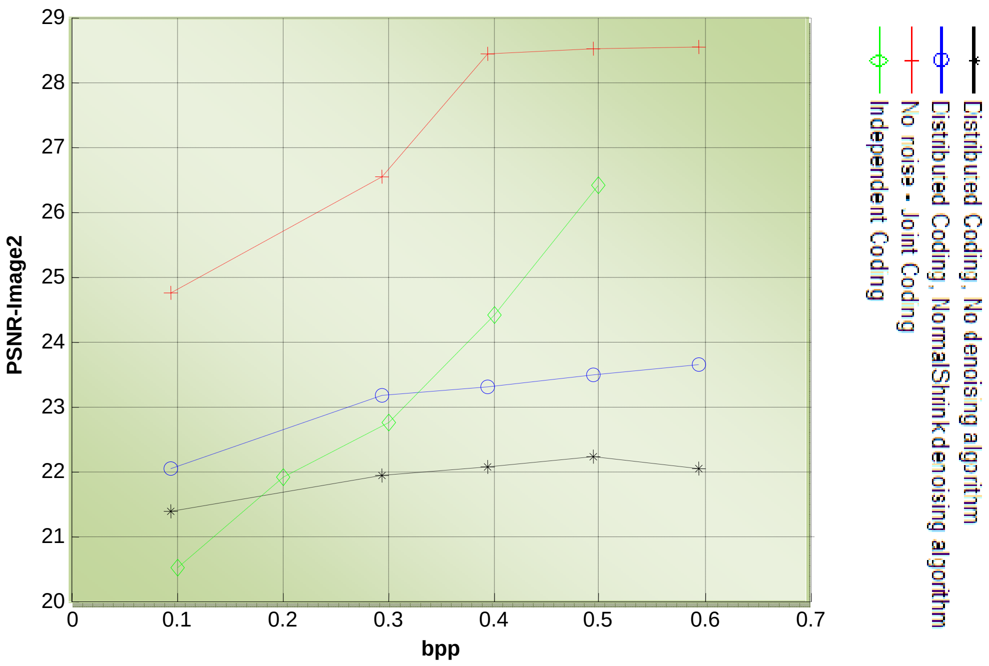


Figure 43 – Comparison of rate-distortion graphs σ = 10, Image1 at 4bpp

In figure 43, the red line shows the joint source coding rate distortion graph. The green line shows the Independent coding rate distortion graph. The black line shows the distributed coding without any denoising on Image1. We included this to see how our denoising algorithm is performing. Finally the blue line shows the distributing coding case with NormalShrink denoising algorithm. We can see that until 0.3bpp the distributed coding rate distortion graph is doing better than the independent coding case and its only 3db below the joint coding case. From 0.3bpp until 0.6bpp the gap increases to 5db and the distributed coding gets very far from the joint coding line.

In figure 44 and 45 we have the same scenario except that Image1 is compressed at 2bpp and 1bpp respectively. Our distributed coding system (blue line, when denoised) is about 6 below the joint coding case and independent coding is doing better than the distributed coding. We can see that the rate which Image1 is compressed at is very important. This is more evident in figure 45, where the independent coding does better than joint coding from 0.5bpp to 0.6bpp.

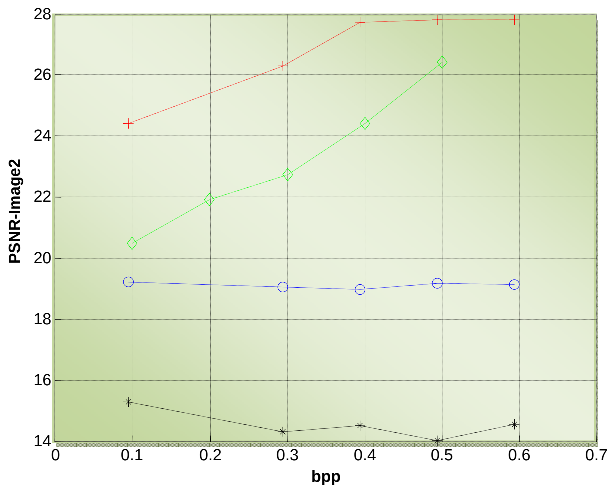


Figure 44 – Comparison of rate-distortion graphs, $\sigma = 10$, Image1 at 2bpp

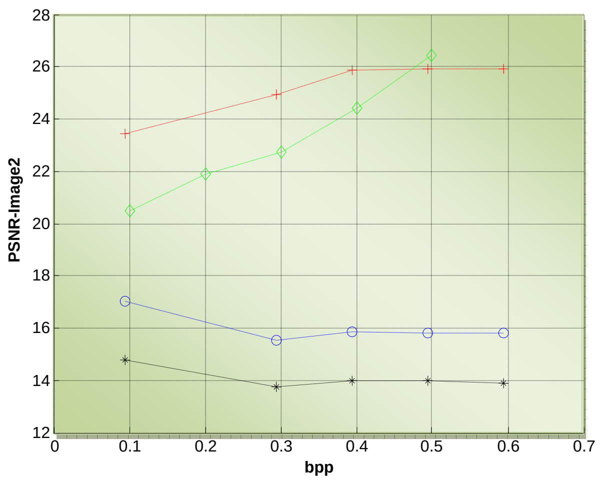


Figure 45 – Comparison of rate-distortion graphs, $\sigma = 10$, Image1 at 1bpp

Now we apply the same tests but with noise standard variation of 20. Figure 46 shows the rate distortion graph of Joint Coding, Distributed Coding and Independent Coding. This time our distributed coding system is 4-5dB below the joint coding case. The independent coding system is doing much better than our distributed coding system.

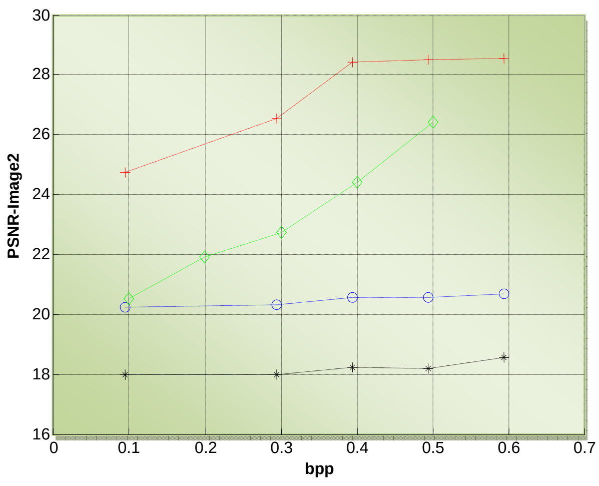


Figure 46 – Comparison of rate-distortion graphs, σ = 20, Image1 at 4bpp

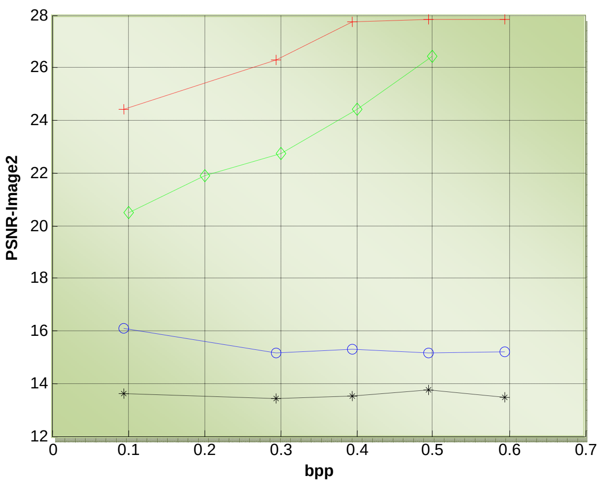


Figure 47 - Comparison of rate-distortion graphs, σ = 20, Image1 at 2bpp

Figures 47 and 48 show the rate distortion graphs at 2bpp and 1bpp for image1 respectively. In figure 47 the blue line i.e. the distributed coding with denoising is not performing well and it even goes below the black line which is the distributed coding without denoising. A possible explanation could be, as stated before, wavelet transform itself is a good denoiser at very low bit-rates as the high frequency components are disregarded.

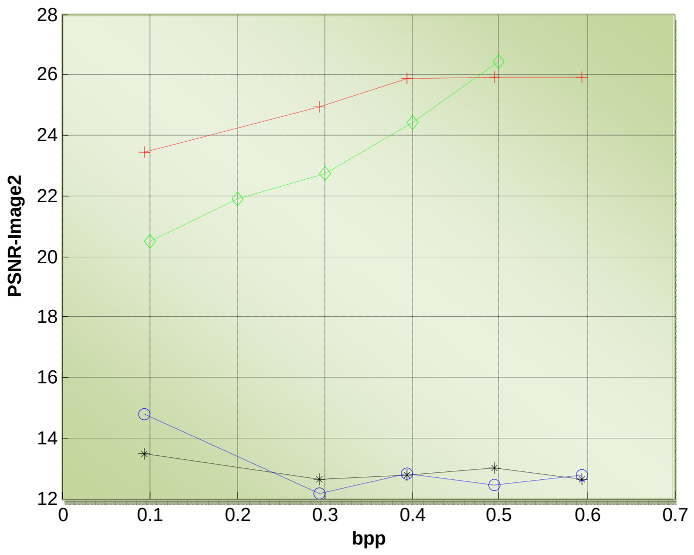


Figure 48 - Comparison of rate-distortion graphs, σ = 20, Image1 at 1bpp

Just for reference, we are going to include the rate-distortion graphs for the case when the standard deviation of noise is 30. We already know that the independent coding case is doing much better than our distributed coding system for the "Room" stereo image pair. Figures 49 up-to 51 show the rate-distortion graphs at 4bpp, 2bpp and 1bpp for image1 respectively.

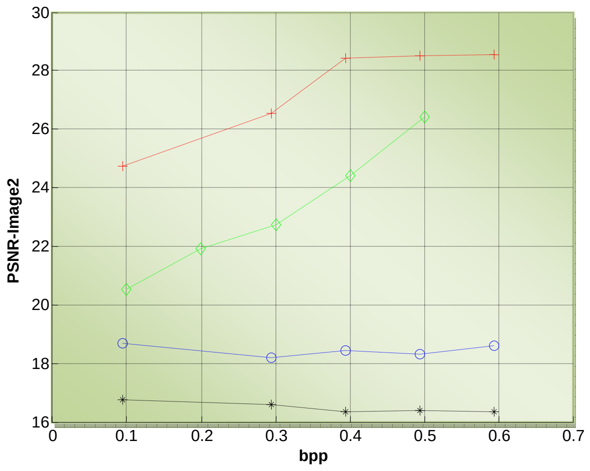


Figure 49 - Comparison of rate-distortion graphs, σ = 30, Image1 at 4bpp

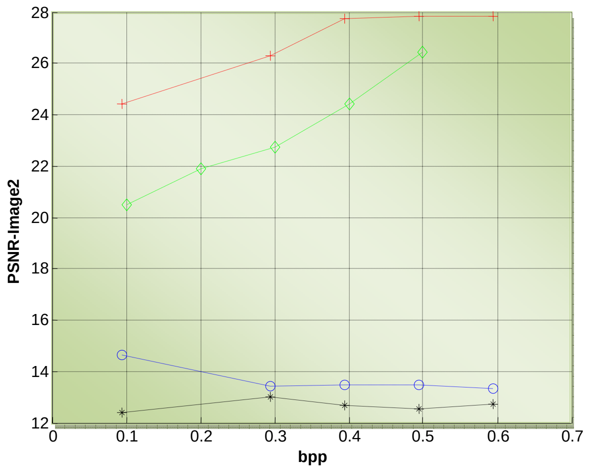


Figure 50 - Comparison of rate-distortion graphs, σ = 30, Image1 at 2bpp

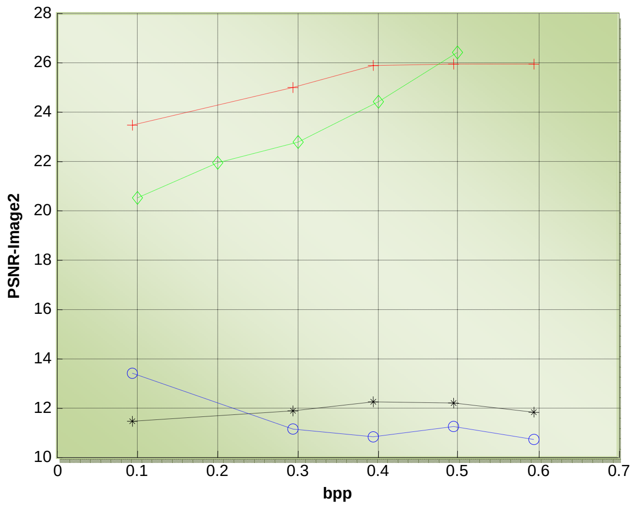


Figure 51 - Comparison of rate-distortion graphs, σ = 30, Image1 at 1bpp

4.2 Stereo Pair 2 – Blocks

Figure 52 and Figure 53 show the original stereo pictures of "Blocks Left" and "Blocks Right" respectively. Figures 54, 55 and 56 show the rate-distortion graphs for Image2 when Image1 is sent to the decoder at the rates 4bpp, 2bpp and 1bpp respectively. In these tests the standard deviation of noise is 10. The corresponding PSNRs of Image1 are 44.04dB at 4bpp, 34.56dB at 2bpp and 27.28dB at 1bpp.

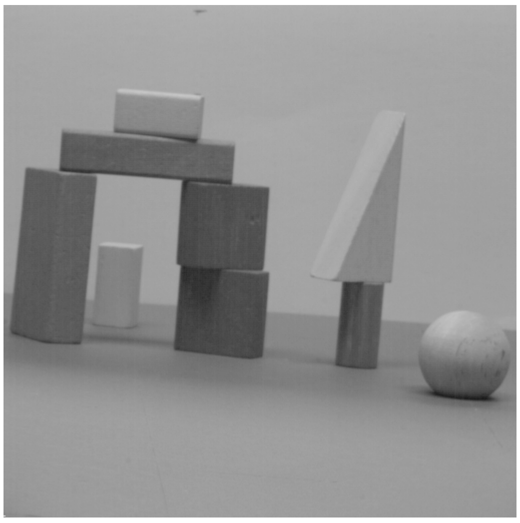


Figure 52 – Original "Block Left" Image

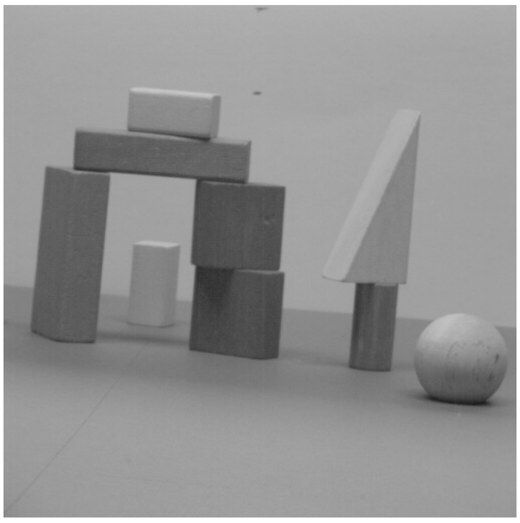


Figure 53 – Original "Block Right" Image

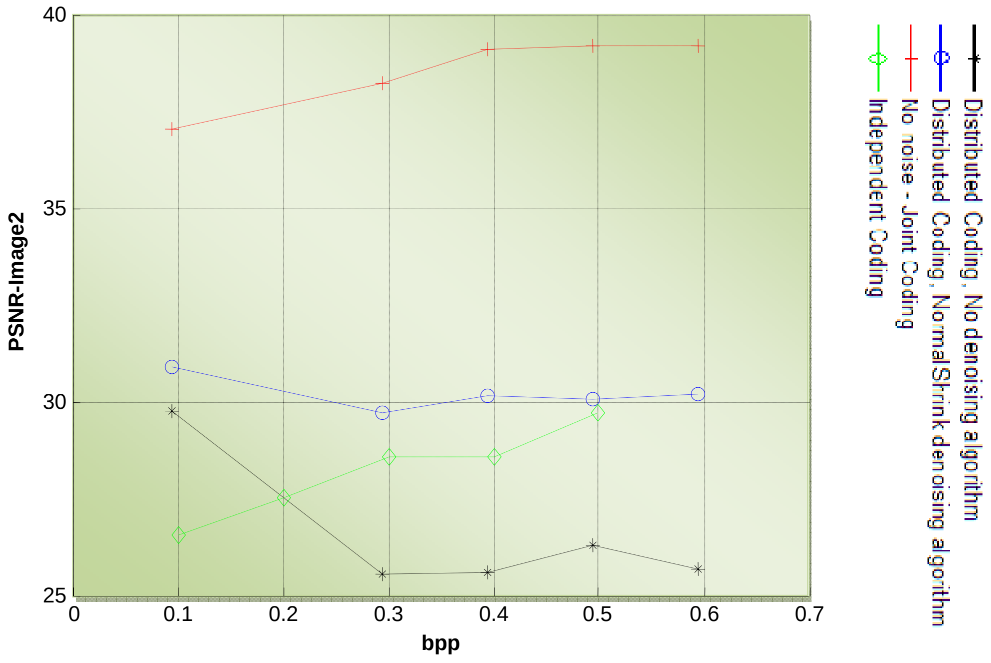


Figure 54 - Comparison of rate-distortion graphs, $\sigma = 10$, Image1 at 4bpp

As can be seen from figure 54 our distributed coding system is not as efficient as the joint source coding case but throughout 0.1-0.6bpp is above the independent coding case. Figure 55 and 56 show the rate distortion graphs at 2bpp and 1bpp for Image1 respectively. We can see from the graphs that, Image1 needs to be coded at the rate of 2bpp or higher in order to get reasonable results for the joint source coding and distributed coding codecs.

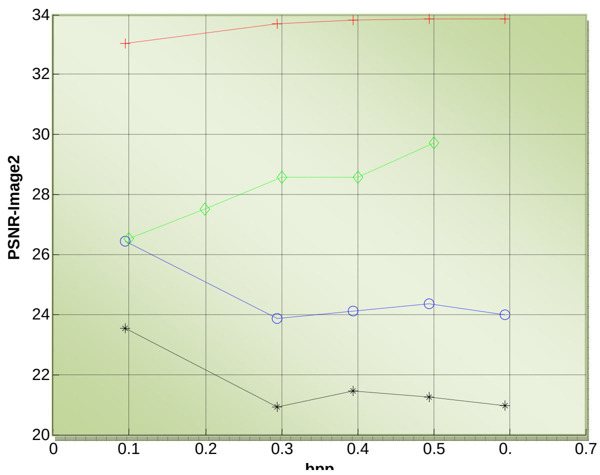


Figure 55 - Comparison of rate-distortion graphs, σ = 10, Image1 at 2bpp

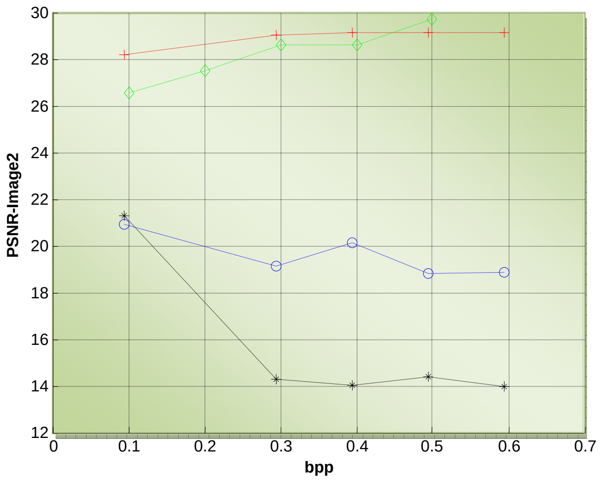


Figure 56 - Comparison of rate-distortion graphs, σ= 10, Image1 at 1bpp

Now we can try the same stereo image pairs with higher noise. We are going to try standard variation of 20 and 30 for the channel noise. Figure 57, 58 and 59 show the rate distortion graphs at the rates 4bpp, 2bpp and 1bpp for image 1 with σ = 20 and figures 60, 61 and 62 show the same graphs with the same specifications, except σ = 30. In figure 58 we can see that our denoising algorithm is not performing well. One possible reason was explained earlier. Another reason could be due to our bit allocation. We have realized that the bit allocation process is not always optimal, especially at low bit rates.

From now on we will only try to compress Image1 at the rates 4bpp and 2bpp. We have seen from our graphs that at the rate of 1bpp, even Independent coding can sometimes do better than the joint coding.

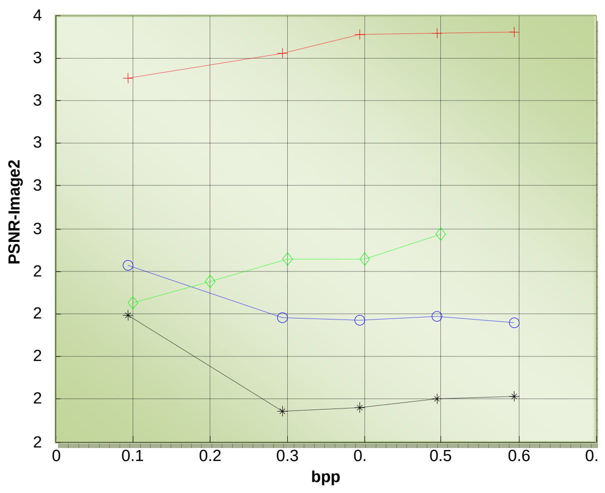


Figure 57 - Comparison of rate-distortion graphs, σ = 20, Image1 at 4bpp

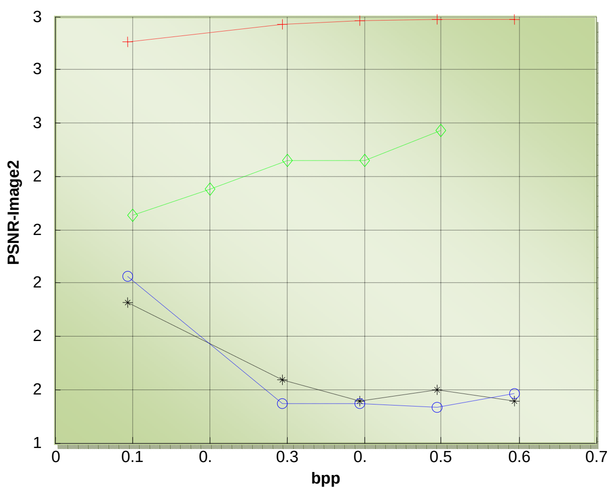


Figure 58 - Comparison of rate-distortion graphs, σ = 20, Image1 at 2bpp

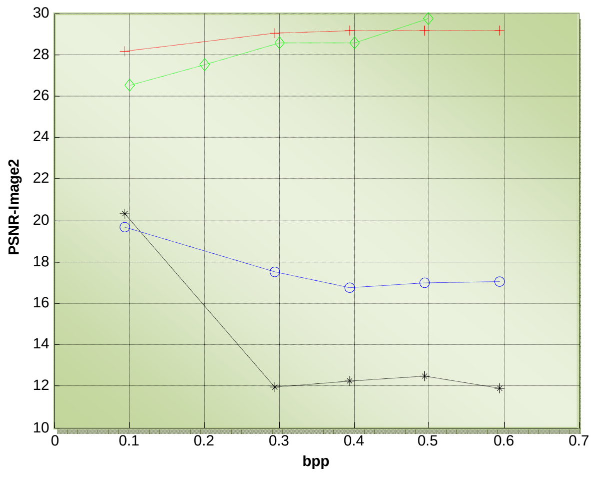


Figure 59 - Comparison of rate-distortion graphs, σ = 20, Image1 at 1bpp

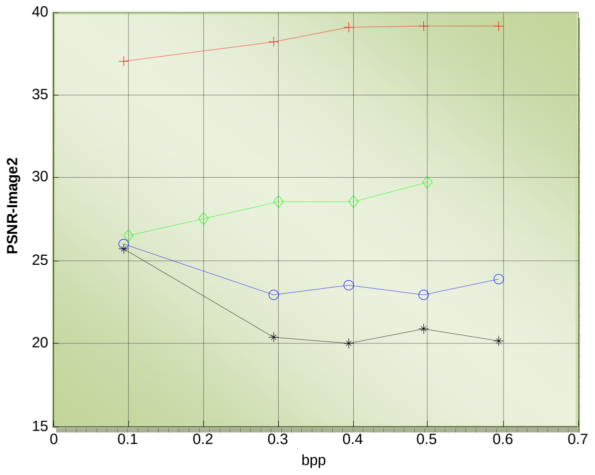


Figure 60 - Comparison of rate-distortion graphs, σ = 30, Image1 at 4bpp

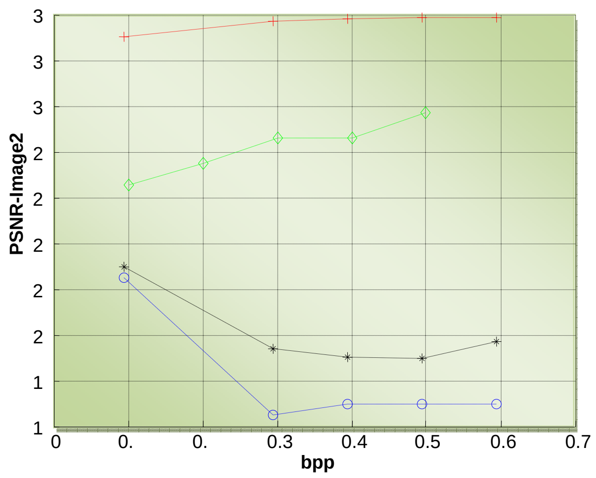


Figure 61 - Comparison of rate-distortion graphs, σ = 30, Image1 at 2bpp

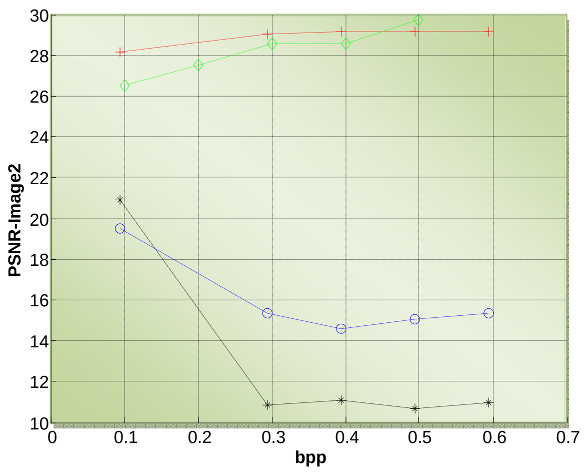


Figure 62 - Comparison of rate-distortion graphs, σ = 30, Image1 at 1bpp

4.3 Stereo Pair 3 – Book

Figures 63 and 64 show the original pictures of “Book Left” and “Book Right” respectively. Figure 65 shows the rate-distortion graphs for Image2 when Image1 is sent to the decoder at the rate of 4bpp. In these tests the standard deviation of noise is 10. The corresponding PSNRs of Image1 are 39.76dB at 4bpp and 32.08dB at 2bpp.

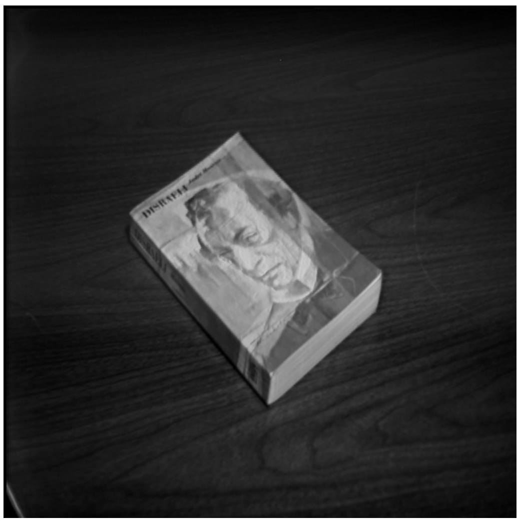


Figure 63 – Original “Book Left” Image

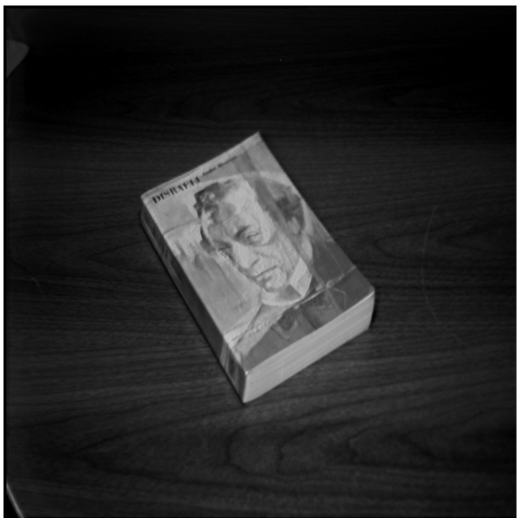


Figure 64 – Original “Book Right” Image

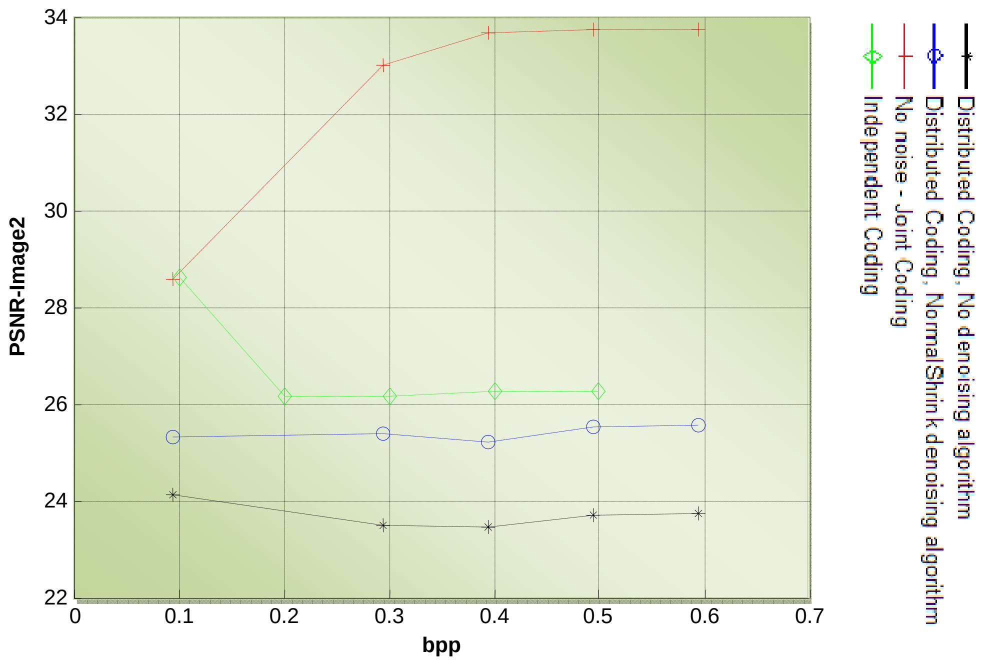


Figure 65 - Comparison of rate-distortion graphs, $\sigma = 10$, Image1 at 4bpp

As can be seen from figure 65 our distributed coding system is not only as efficient as the joint source coding case but also throughout 0.1-0.6bpp is below the independent coding case. Also we can see that with the independent coding case the maximum PSNR happens at 0.1bpp. This shows that our bit allocation process is not optimal. Figures 66, 67, 68 and 69 show the rate distortion graphs for $\sigma = 20$ and $\sigma = 30$ at the rates of 4bpp and 2bpp for Image1. We can see that our distributed coding system is always well below the Joint coding and Independent coding.

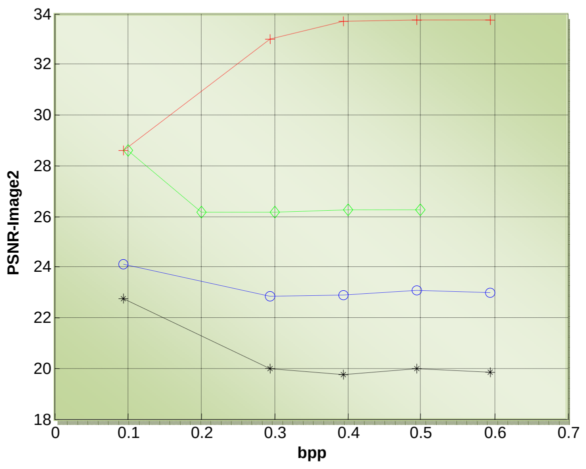


Figure 66 - Comparison of rate-distortion graphs, σ = 20, Image1 at 4bpp

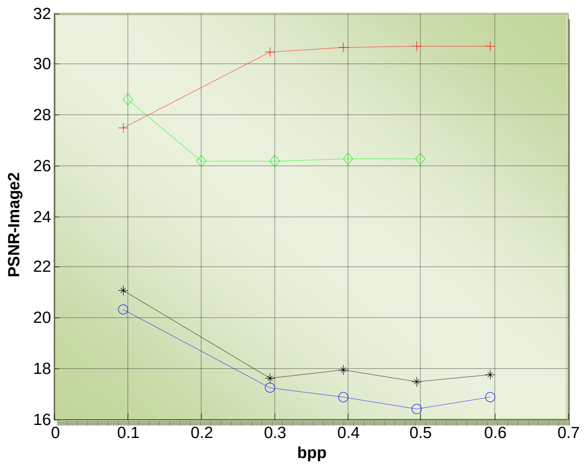


Figure 67 - Comparison of rate-distortion graphs, σ = 20, Image1 at 2bpp

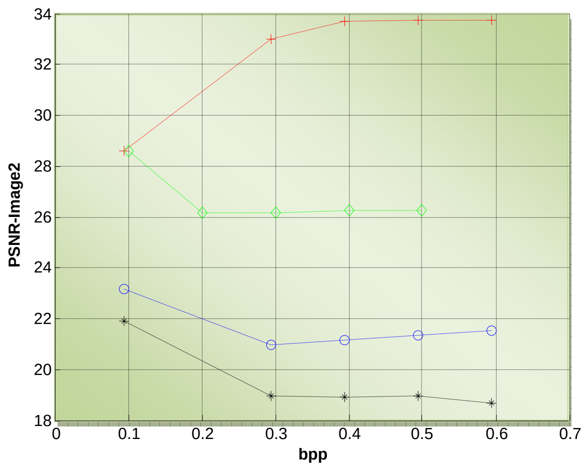


Figure 68 - Comparison of rate-distortion graphs, σ = 30, Image1 at 4bpp

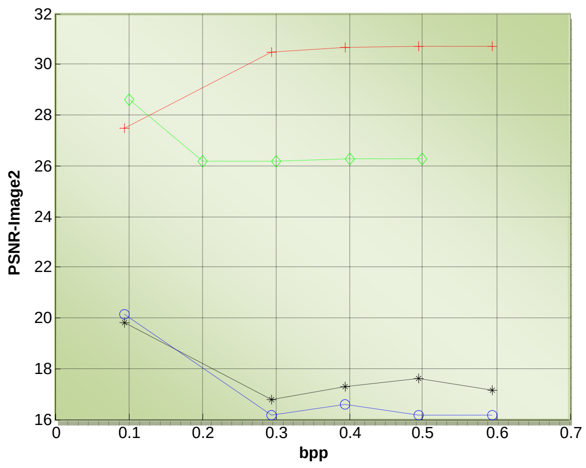


Figure 69 - Comparison of rate-distortion graphs, σ = 30, Image1 at 2bpp

We can see that overall we get poor results for distributed coding rate-distortion graphs.

4.4 Stereo Pair 4 – Spider

Figures 70 and 71 show the original stereo pictures of “Spider Left” and “Spider Right” respectively. Figures 72 and 73 show the rate-distortion graphs for Image2 when Image1 is sent to the decoder at the rates 4bpp and 2bpp. In these tests the standard deviation of noise is 10. The corresponding PSNRs of Image1 are 38.87dB at 4bpp, 32.64dB at 2bpp.

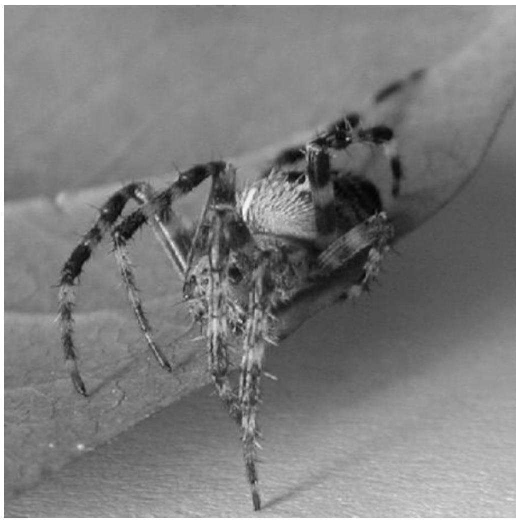


Figure 70 – Original “Spider Left” Image

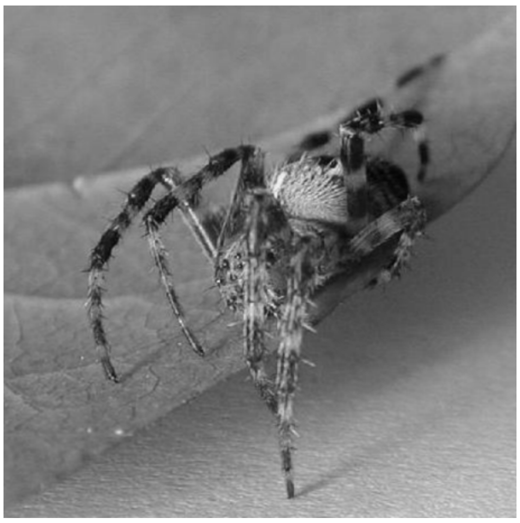


Figure 71 – Original “Spider Right” Image

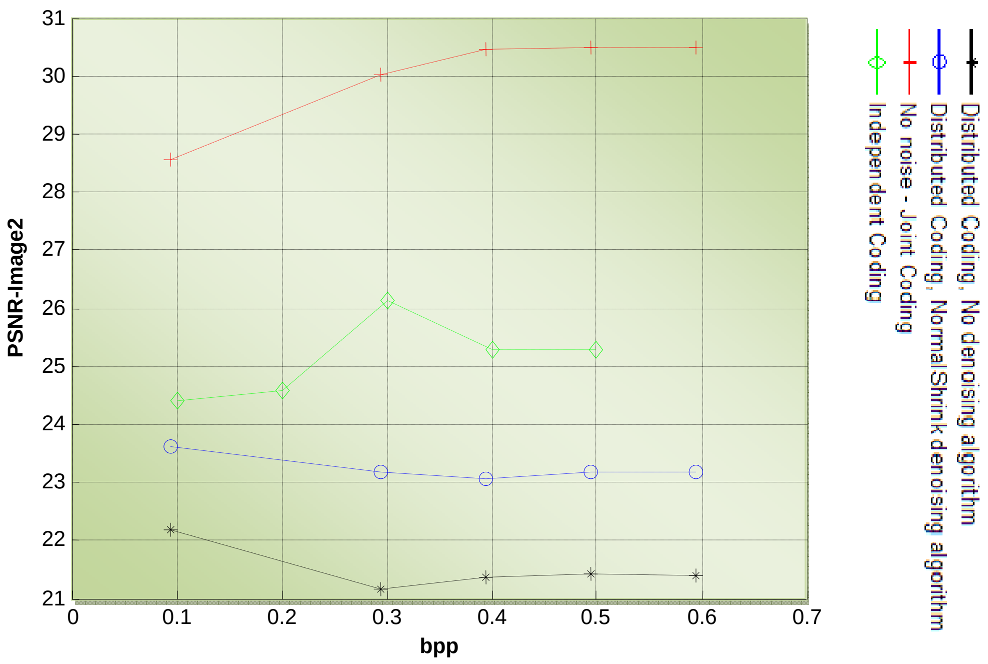


Figure 72 - Comparison of rate-distortion graphs, σ = 10, Image1 at 4bpp

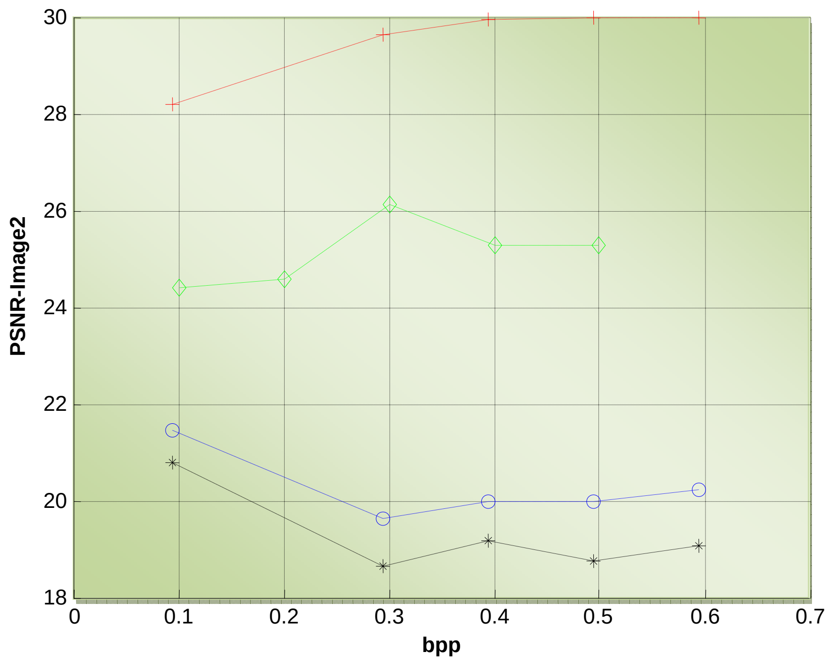


Figure 73 - Comparison of rate-distortion graphs, σ = 10, Image1 at 2bpp

From figure 72 we can see that the distributed coding rate-distortion graph is close to the independent coding case and is about 5 to 7db below the joint coding case. Figure 73 shows the same comparison but at the rate of 2bpp for Image1. The rate-distortion graph of distributed coding is well below the Independent coding and Joint Coding. This shows how important it is in the way we send Image1 to the decoder.

4.5 Stereo Pair 5 – Tennis_Ball

Figures 74 and 75 show the original pictures of “Tennis_Ball Left” and “Tennis_Ball Right” respectively. Figures 76 and 77 show the rate-distortion graphs for Image2 when Image1 is sent to the decoder at the rates 4bpp and 2bpp. In these tests the standard deviation of noise is 10. The corresponding PSNRs of Image1 are 46.52dB at 4bpp, 39.72dB at 2bpp.

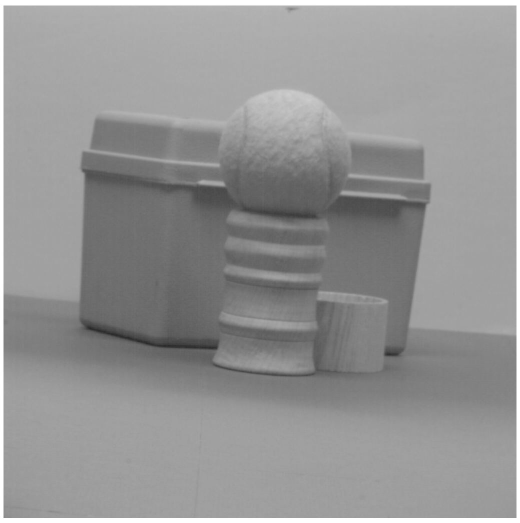


Figure 74 – Original “Tennis_Ball” Left

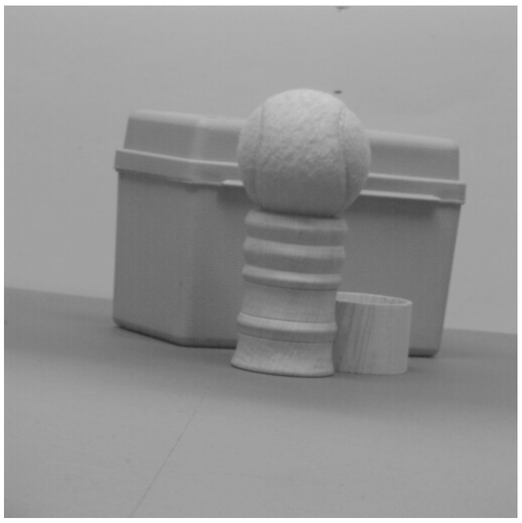


Figure 75 – Original “Tennis_Ball” Right

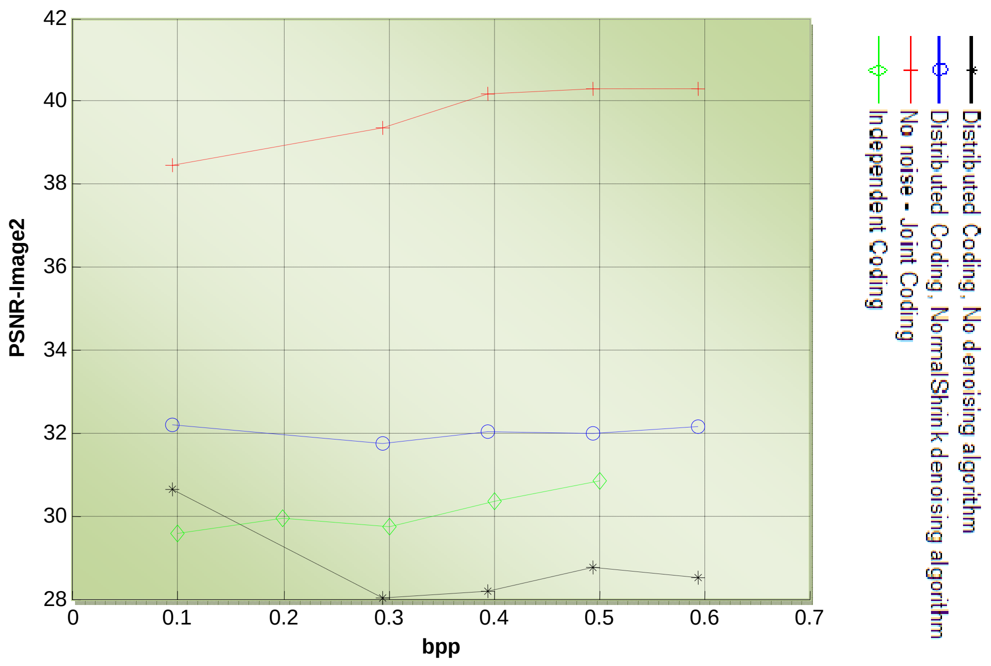


Figure 76 - Comparison of rate-distortion graphs, σ = 10, Image1 at 4bpp

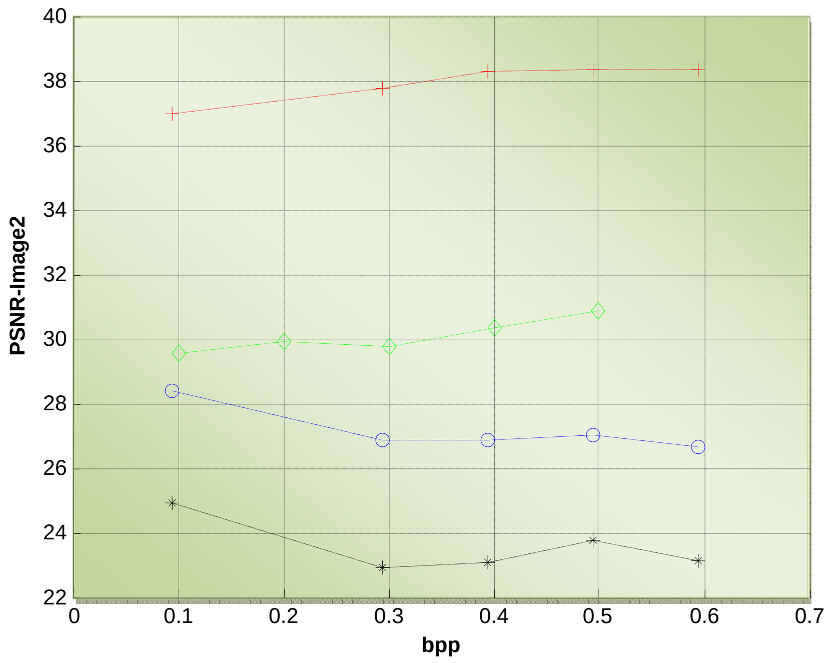


Figure 77 - Comparison of rate-distortion graphs, σ = 10, Image1 at 2bpp

As show on figure 76 the rate distortion graph for distributed coding is well above the Independent coding throughout the 0-0.6bpp and is 6-8dB below the joint coding case. In figure 77 where, Image1 is sent at 2bpp, the rate distortion graph for distributed coding is below the independent coding by 1-4dB. This again shows how important it is in the way we send Image1 to the decoder. Figures 78 and 79 show the rate distortion graphs for the cases when $\sigma = 20$ and $\sigma = 30$ respectively. Image1 for both cases is compressed at 4bpp.

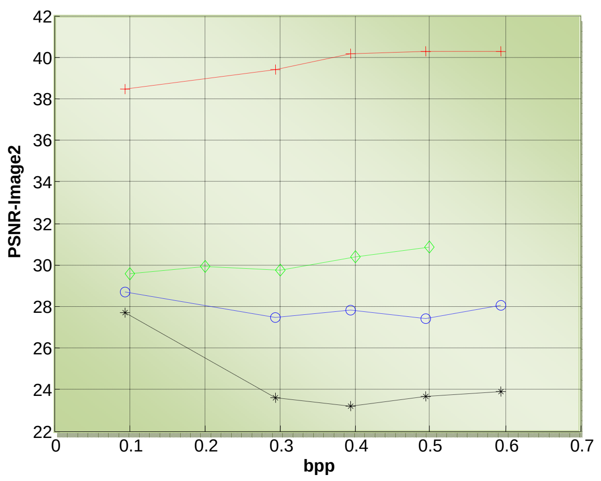


Figure 78 - Comparison of rate-distortion graphs, σ = 20, Image1 at 4bpp

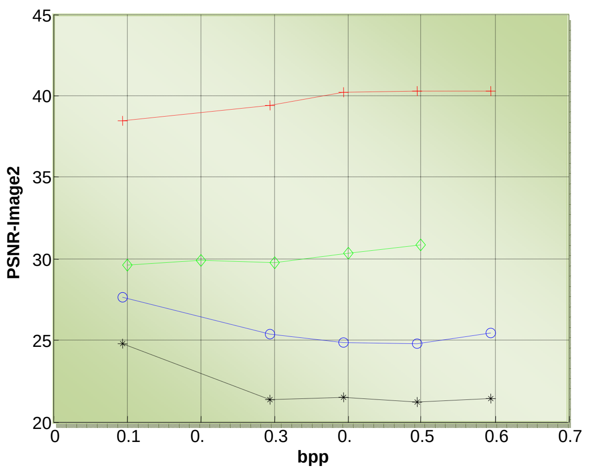


Figure 79 - Comparison of rate-distortion graphs, σ = 30, Image1 at 4bpp

Chapter 5

Further Improvements and Future Work

The aim of this project is to see if distributed coding is as efficient as joint coding when we have a noisy channel between encoder1 and encoder2. Up until now we get poor results for the distributed coding rate distortion graphs. We have proved that the stereo pictures itself, the bit-rate that image1 is compressed at, and the variance of the noise, play an important part on the results we achieve. Now we are going to make some modifications to see if the performance of the rate-distortion graphs of distributed coding get any better.

5.1 Denoising

We used Haar filters when we were denoising the corrupted Image1 in encoder2. We could also use Daubechies filters for denoising corrupted signals. In order to try denoising using Daubechies filters we first need to find the inverse wavelet transform of the corrupted image in the encoder with the use of Haar filters. This is because for the compression of Image1 in encoder1 we used Haar filters, so we have to use Haar filters again to take the inverse wavelet transform. Now we take the wavelet transform of the Image using Daubechies filters, denoise it, and then take the inverse wavelet transform with the same filters. The choice of which Daubechies filters to be used is quite important but generally 'db8' (8 vanishing moments) filters work quite well. Also the choice of number of decompositions is important but generally 3-level decompositions are preferred for 512x512 images.

It is important to mention here that we have an image border distortion problem with Daubechies filters in MATLAB. There are different ways of handling the distortion problem using the MATLAB "dwtmode" function. The function "dwtmode" specifies the image extension mode for discrete wavelet transform. Some of the modes are "sym" for symmetric-padding (half-point), "zpd" for zero-padding and "ppd" for periodic-padding. Zero-padding usually gives the best result for our test images so we stick to "zpd" mode when using Daubechies filters for denoising. We will show some examples on how denoising under ‘db8’ with 3-level wavelet decomposition performs by comparing to the Haar filters case with 5-level decomposition.

Figure 80 and Figure 81 show the denoised “Room” image with the use of Haar and ‘db8’ filters respectively. The standard variation of noise was chosen to be 40. The corresponding values of PSNRs are 23.3dB and 24.4dB respectively. We can see that with the use of ‘db8’ filters we can improve our denoised image by about 1dB. We can try another image. Figure 82 and Figure 83 show the denoised “Spider” image with the use of Haar and ‘db8’ filters respectively. The standard variation was chosen to be 40. The corresponding values of PSNRs are 24.86dB and 25.31dB respectively. Therefore with ‘db8’ filters we improved our denoised image by about 0.5dB.

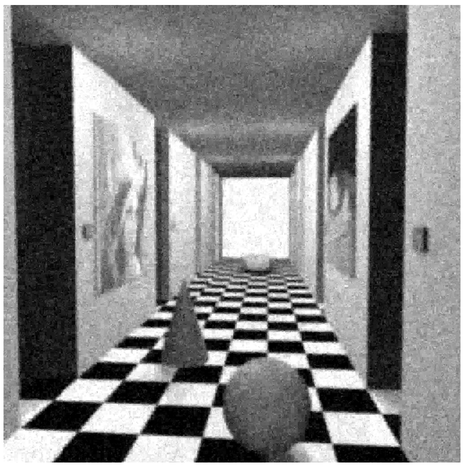


Figure 80 – Denoise Image using Haar filters

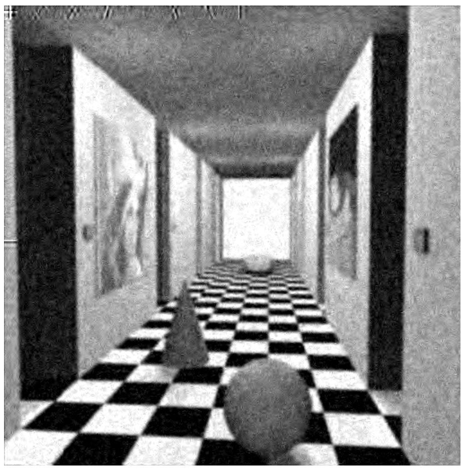


Figure 81 – Denoised Image using 'db8' filters

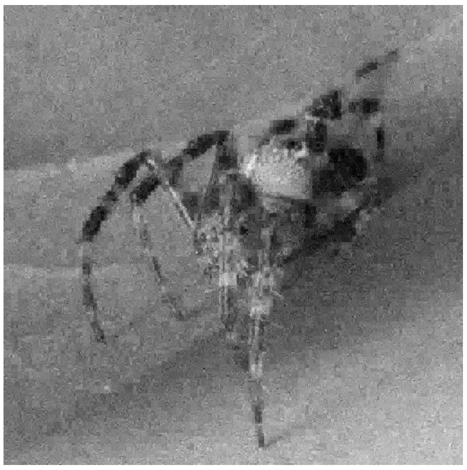


Figure 82 - Denoise Image using Haar filters

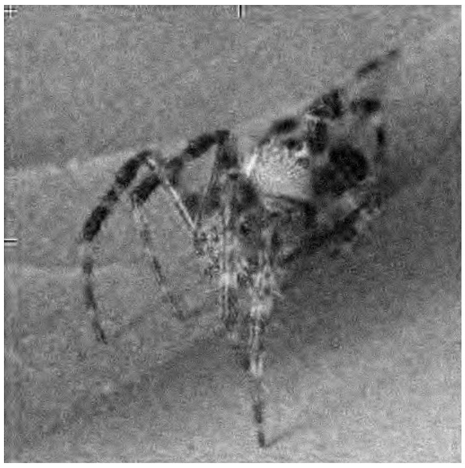


Figure 83 - Denoised Image using 'db8' filters

Now we show some examples of rate-distortion graphs with the use of 'db8' filters and compare it our previous results where we were using Haar filters for denoising the corrupted Image1 in encoder2. Figure 84 and Figure 85 show the rate distortion graphs for "Book" stereo image pairs. In Figure 84 we are using Haar filters to denoise Image1 and in Figure 85 we are using 'db8' filters to denoise Image1. As can be seen from the figures, the distributed coding (blue line) rate distortion graph has gone up by about 1db when we use 'db8' filters. Another example is shown in Figure 86 and Figure 87 for the "Tennis_Ball" stereo pair. As can be seen from the figures the distributed coding rate distortion graph with the use of 'db8' filters is about 2dB higher.

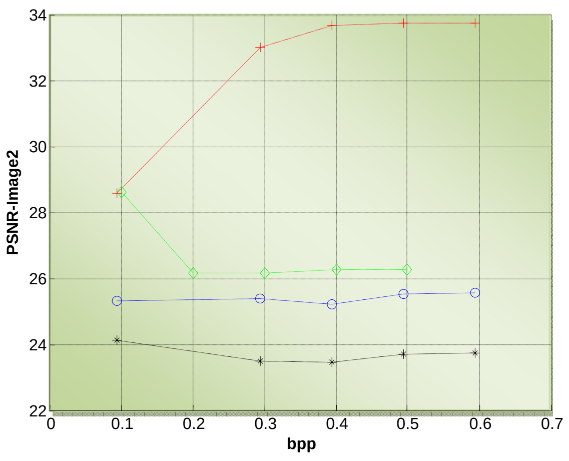


Figure 84 - Rate-distortion graphs, $\sigma = 10$, Image1 at 4bpp, Haar filters

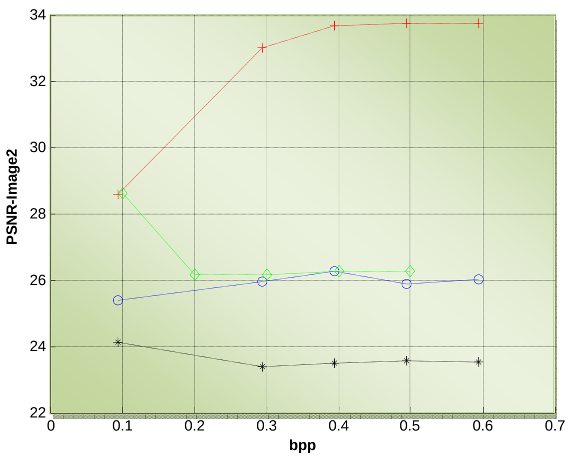


Figure 85 - Rate-distortion graphs, $\sigma = 10$, Image1 at 4bpp, 'db8' filters

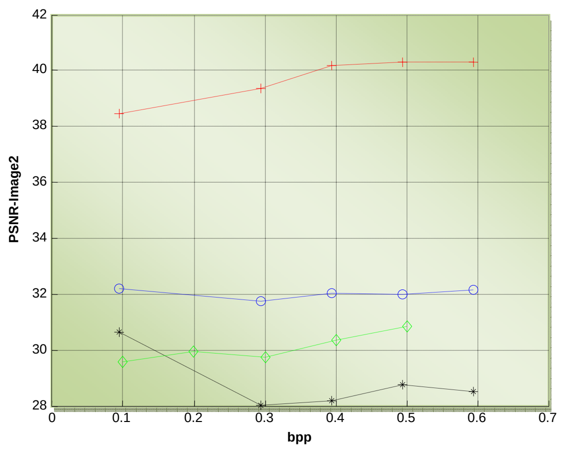


Figure 86 - Rate-distortion graphs, $\sigma = 10$, Image1 at 4bpp, Haar filters

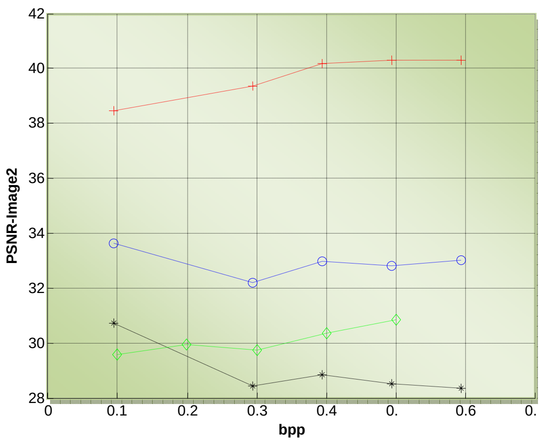


Figure 87 - Rate-distortion graphs, $\sigma = 10$, Image1 at 4bpp, 'db8' filters

5.2 Block Matching

The method we used for our motion compensation is called fixed block matching. Fixed block matching means that the block sizes are fixed to KxK. Fixed block matching is very popular due its simplicity. Other block matching methods are "Variable Block Matching" and "Overlapped Block Matching". Generally theses two perform much better than fixed block matching at normal bit-rates. At low bit rates fixed block matching performs better than the other two. This is because, with fixed block size method we can pre-specify the block size for both the encoder and the decoder so no other information is needed to be sent to the decoder about the block size. With "Variable Block Matching" method this is not case and we need to send the variable block sizes to the encoder for each captured image. Also, for overlapped block matching method, there are more bits needed to send the motion vector to the decoder. Therefore at low bit-rates "Fixed Block Matching" is probably the best available method for motion compensation.

5.3 Bit Allocation

As mentioned before, from our results we realized that the bit allocation algorithm we use is not optimal. It may be impossible to find an optimal bit allocation algorithm which would optimally work on very low bit-rates. In JPEG2000 codec system, a weighted bit allocation is used instead. In this way, the lowest weights are given to the diagonal subbands and the highest weight is given to the approximation subband. Figure 88 shows one way of setting the weights for different subbands. Now we will show how this would improve our rate-distortion graphs significantly by showing some examples.

Figure 89 shows the rate-distortion graph of "Tennis_Ball" stereo pair with the following specifications: σ =10, rate of Image1 = 4bpp, and 'db8' filters for denoising with the non-weighted bit allocation algorithm. Figure 90 shows the same rate distortion graph with the weighted bit allocation algorithm. The results are very surprising. The rate distortion graph for our distributed coding system has gone up by at least 4db. Now we can even say that our distributed coding is very close to the joint coding case. This shows how important the bit allocation process is. Now we can change the specifications and compare the new results to the previous results where non-weighted bit allocation algorithm was used.

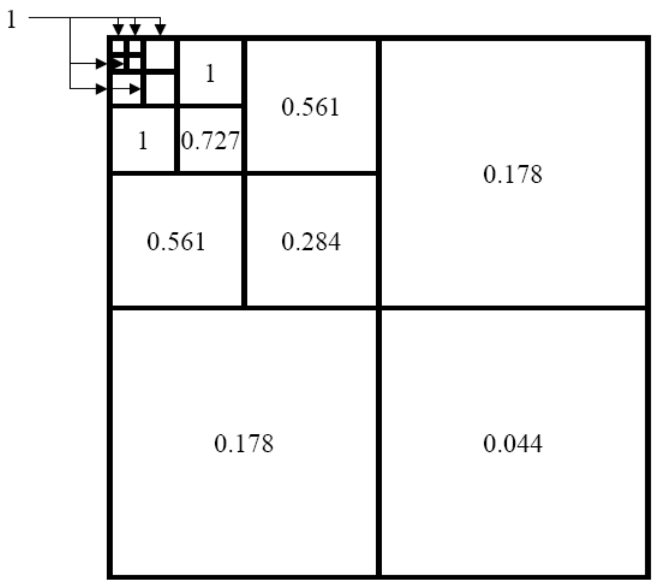


Figure 88 – Possible weighting table for bit allocation

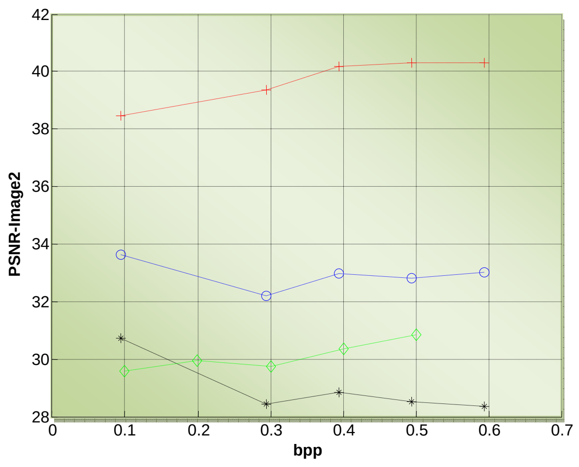


Figure 89 – Rate-distortion graphs, σ = 10, Image1 at 4bpp, 'db8' filter, Non-Weighted Bit Allocation

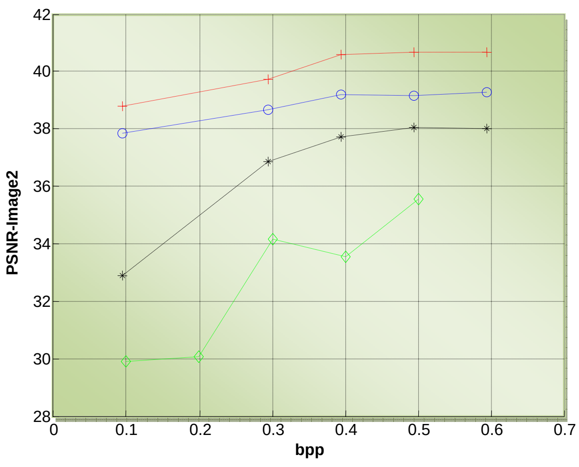
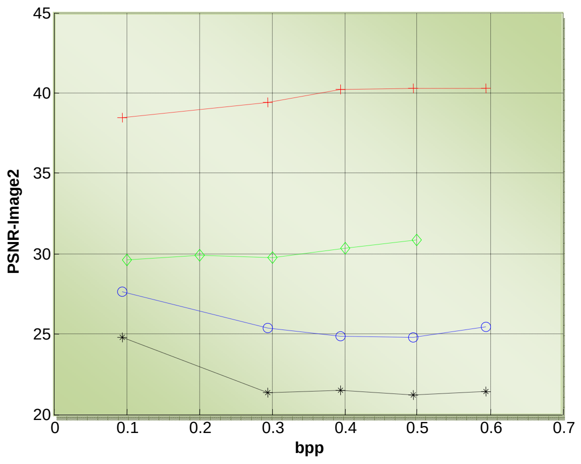


Figure 90 – Rate-distortion graphs, σ = 10, Image1 at 4bpp, 'db8' filter, Weighted Bit Allocation

Figure 91 and Figure 92 show the rate-distortion graphs of the "Tennis_Ball" stereo pair with the non-weighted bit allocation algorithm and the weighted bit allocation algorithm respectively. The specifications for both cases are as follows: σ =30, rate of Image1 = 4bpp, 'db8' filters for denoising. Again the results have improved significantly with the weighted bit allocation. Even at high noise variances the distributed coding rate distortion graph is doing as good as the rate distortion graph of independent coding scenario. One important thing to mention here is that even the rate distortion graph of Independent coding has improved but the change in distributed coding case is much higher.



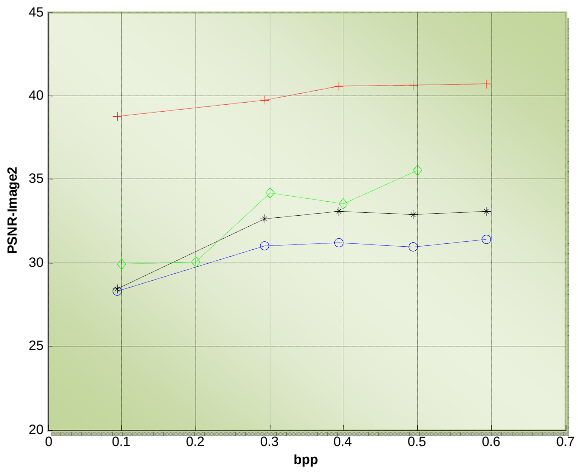
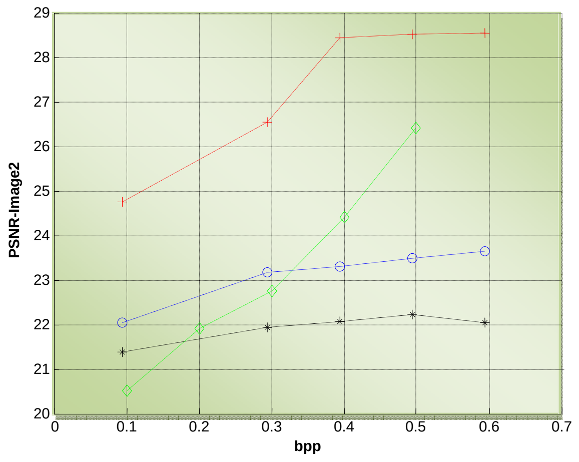


Figure 91 - Rate-distortion graphs, σ = 10, Image1 at 4bpp, 'db8' filter, Non-Weighted Bit Allocation

Figure 92 - Rate-distortion graphs, σ = 30, Image1 at 4bpp, 'db8' filter, Weighted Bit Allocation

Figure 93 and Figure 94 show the rate-distortion graphs of the "Room" stereo image pair with the non-weighted bit allocation algorithm and the weighted bit allocation algorithm respectively. The specifications for both cases are as follows: σ =10, rate of Image1 = 4bpp, 'db8' filters for denoising. This time the results for distributed coding have actually gone down. This shows the importance of the weighting table. An adaptive weighted bit allocation algorithm could be an immediate future work for this project.



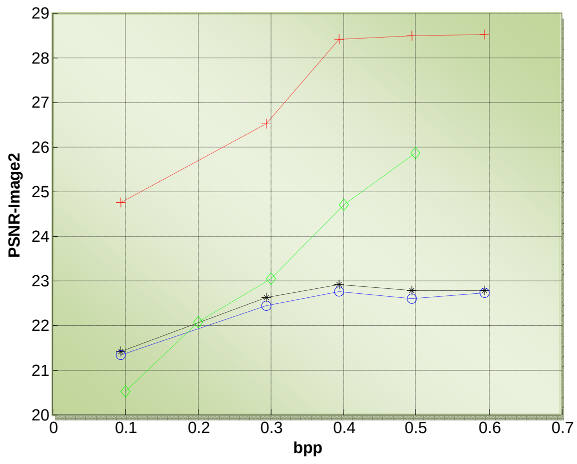


Figure 93 - Rate-distortion graphs, σ = 10, Image1 at 4bpp, 'db8' filter, Non-Weighted Bit Allocation

Figure 94 - Rate-distortion graphs, σ = 10, Image1 at 4bpp, 'db8' filter, Weighted Bit Allocation

5.4 Residual Coding

We use wavelet transform to compress our residual image in the second encoder. In [17] Moellenhiff's analysis shows that residual images show significantly different characteristics from ordinary images. The pixels of the residual images are less correlated when compared to the original captured image and they mostly contain edges and high frequency information. This would mean that, wavelet transform which works very well on compressing images would not very useful for residual images. Also, as mentioned before, the range of residual images is most likely from -255 to 255 for 8-bit images therefore uniform quantization would not be very useful for residual coding. One way to tackle the problem is to take the discrete cosine transform (DCT) of the disparity blocks that are below a threshold and take the wavelet transform of the disparity blocks that are above the set threshold. In [18] a similar method was developed by Frajka and Zeger and they achieve some good results. Improvements on residual coding could be a future work for this project.

Summary and Conclusion

The aim of this project was to see if distributed coding is as efficient as joint coding when we have a noisy channel between encoder1 and encoder2. It was shown that the stereo pictures themselves, the bit-rate that image1 is compressed at, and the variance of the noise, play an important part on the rate-distortion graphs. If the stereo pictures are highly detailed then we need to choose a smaller block size for motion compensation. This would mean more data needs to be sent to the decoder and generally is not a good solution at low bit-rates. The bit-rate that Image1 is compressed at is extremely important, especially if the variance of the channel noise is high. If the reference image is highly compressed, then due to the noise in the channel, it is extremely difficult to achieve an acceptable quality for the denoised Image1 at the second encoder. Also if the variance of the channel noise is too high then it would be difficult to be even as efficient as independent coding.

The choice of filters for the wavelet transform is quite important as well especially for denoising images. We showed that by using 'db8' filters instead of Haar filters we can, on average, improve the PSNR of our denoised image by about 1db. We also showed how extremely important the process of bit allocation is, especially when we are working at low bit-rates. The weighted bit allocation improved some of the rate-distortion graphs dramatically. A probable immediate future work for this project would be to design a new adaptive weighted block matching algorithm which would work on most images. To conclude, we can say that although being as efficient as joint source coding with distributed coding is practically impossible, but on certain conditions we can get very close to it.

Bibliography

[1] D. Slepian and J. K. Wolf, ``Noiseless coding of correlated information sources,'' IEEE Trans. Inform. Theory, vol. IT-19, pp. 471-480, July 1973.

[2] A. D. Wyner and J. Ziv, ``The rate-distortion function for source coding with side information at the decoder,'' IEEE Trans. Inform. Theory, vol. IT-22, pp. 1-10, January 1976.

[3] A. D. Wyner, ``On source coding with side information at the decoder, IEEE Trans. Inform. Theory, vol. IT-21, pp. 226-228, May 1975.

[4] A.D. Wyner, "The rate distortion function four source coding with side information at the decoder-II: General sources," Information and Control, vol. 38, pp. 60-80, May 1978.

[5] B. Girod, A. Aaron, S. Rane and D. Rebollo-Monedero , "Distributed video coding," Proceedings of the IEEE, Special Issue on Video Coding and Delivery, vol. 93, no.1, pp. 71-83, Jan. 2005.

[6] K. Varma and A. Bell, ”JPEG2000-Choices and Tradeoffs for Encoders”, IEEE Signal Processing Magazine, pp.70-74, November 2004.

[7] A. Skodras, C. Christopoulos and T. Ebrahimi, "The JPEG2000 still image compression standard", IEEE Signal Processing Magazine, vol. 18, no. 5, pp. 36–58, September 2001.

[8] H. Wu and A. Abouzeid, “Energy efficient distributed image compression in resource constrained multihop wireless networks”, Computer Communications, vol. 28, pp. 1658-1668, September 2005

[9] M. Flierl and P.Vandergheynst, “Distributed Coding of Highly Correlated Image Sequences with Motion-Compensated Temporal Wavelets”, EURASIP Journal on Applied Signal Processing, vol. 2006, id 46747, 2006.

[10] R. Wagner, R.D. Nowak and R.G. Baraniuk, “Distributed image compression for sensor networks using correspondence analysis and super-resolution”, In Proceedings of ICIP 2003, pp. 597-600, 2003.

[11] O. Rioul and M. Vetterli, “Wavelets and Signal Processing,” IEEE Signal Processing Magazine, vol. 8, no. 4, pp. 14-38, October 1991.

[12] [Book] – A. Abbate, C.M. DeCusatis and P.K. Das, “Wavelets and Subbands fundamentals and applications”, September 2001.

[13] A. Ortega, "Optimal rate allocation under multiple rate constraints'', Data Compression Conference, March 1996.

[14] E. A. Riskin, "Optimum bit allocation via the generalized BFOS algorithm,'' IEEE Trans. Inform. Theory, vol. 37, pp. 400-402, March 1991.

[15] L.Kaur, Savita Gupta and R.C.Chauhan, “Image Denoising using Wavelet Thresholding”, Third Conference on Computer Vision, Graphics and Image Processing, December 2002.

[16] E. A. Edirisinghe, M. Y. Nayan and H. E. Bez, “A wavelet implementation of the pioneering block-based disparity compensated predictive coding algorithm for stereo image pair compression”, Signal Processing: Image Communication, vol. 19, Issue 1, pp. 37-46, January 2004,

[17] M. S. Moellenho and M. W. Maier, “Characteristics of disparity-compensated stereo image pair residuals", Signal Processing: Image Communication, vol. 14, pp. 55-69, 1998.

[18] A. Frajka and K. Zeger “Residual image coding for stereo image compression”, International Conference on Image Processing, vol. 2, pp. II-220 and II-271, vol. 2, January 2003

Appendix

A CD-ROM is also available at the end report with all the MATLAB codes, the stereo pictures used and the PDF document of this report.

“Encoder 1”

```
function [T M gamma] = encoder1(image1,rate,h);
% function endoder1 compresses Image1 at the given bit-rate. "h" is used
for
% enabling or disabling quantization. The outputs are T, the transformed
% image, M, the values of minima and maxima, gamma the calculated bits for
% different subbands using the bitalloc function

% Reading Image1
X1 = readimage(image1);

% Taking the wavelet transform of the image with N decompositions,
% WaveQuant function also does the bit allocation and quantization for all
% the subbands
[T M gamma] = WaveQuant(X1,5,'db1',rate,0,h);
```

“Encoder 2”

```
function [vert_disp,horz_disp,par,M_par,gamma2] =
encoder2(image2,sigma,X1,M,gamma,rate2,k,v);
% The encoder2 function receives a noisy version of Image1. It first has to
% denoise Image1. Then using motion compensation, it estimates Image2 from
% Image1. The residual image is found by subtracting Image2_estimated from
% the original Image2. Finally the residual image is coded using the
% Wavequant function. k is a flag used for enabling/disabling the denoising
% process. Sigma is the standard variation of the noise. v is used for
% enabling/disabling quantization.

% Receive the noisy Image1
X1 = channel_noise(sigma,X1);

% Ignore the the subbands B1, C1, D1, B2, C2 and D2. This is very useful
% for low bit-rates, when denoising
if k==1
    gamma = [zeros(1,6) gamma(7:16)];
end

% WaveDeQuant does Inverse Quantization and inverse wavelet transform with
% the option of denoising
X1 = WaveDeQuant(X1,M,gamma,5,'db1',0,0,v);

X1 = uint8(X1);
X1 = double(X1);

% The denoising algorithm is applied in the WaveDeQuant function
if k==1
    [X1 Mtmp gammatmp] = WaveQuant(X1,3,'db8',0,0,0);
    X1 = WaveDeQuant(X1,Mtmp,gamma,3,'db8',sigma,1,0);
end
X1 = uint8(X1);
```

```
X1 = double(X1);
X2 = readimage(image2);
cmr=0; flag=0;
b = 8;  % block_size
T = 512/b; % Motion vector size
w = 9;  % search area (surrounding blocks) has to be an odd number
w_max = (w-1)/2;
horz_disp = zeros(T,T);
vert_disp = zeros(T,T);
for m = 1:1:T
    c = ((m-1)*b)+1;
    for n = 1:1:T
        L = inf;
        d = ((n-1)*b)+1;
        ref = X2(c:c+(b-1),d:d+(b-1)); % A Block is Image2 is considered
        for i = 1:1:w
            for j = 1:1:w
                e = c + ((i-w_max-1)*b);
                g = e + (b-1);
                f = d + ((j-w_max-1)*b);
                h = f + (b-1);
                if (e>0) & (f>0) & (g>0) & (h>0) & (g<513) & (h<513)
                    mean = sum(sum(abs(X1(e:g,f:h) - ref)));
                    if mean < L
                        L = mean;
                        col = j-w_max-1;
                        row = i-w_max-1;
                    end
                end
            end
        end
        horz_disp(m,n) = col;
        vert_disp(m,n) = row;
        Q = c+b*vert_disp(m,n);
        W = d+b*horz_disp(m,n);

        % Making Image2 estimated version from the horizontal and
        % vertical displacements
        X2_est(c:c+(b-1),d:d+(b-1)) = X1(Q:Q+(b-1),W:W+(b-1));
    end
end
par = (X2-X2_est); % The residual Image

% We force the range to be within -128 and 128
for m=1:512
    for n=1:512
        if abs(par(m,n)) >= 128
            par(m,n) = sign(par(m,n))*128;
        end
    end
end

% Compressing the residual Image
[par M_par gamma2] = WaveQuant(par,5,'db1',rate2,1,v);
```

"Decoder"

```
function decoder(image1,image2,sigma,rate3,h)
% This is the main program. We first set the two stereo images with the
% standard deviation of the channel noise and the rate at which we want to
% compress Image1. h here is used for enabling/disabling the quantizer.

% Setting the wavelet extension mode to zero-padding
dwtmode('zpd');
clc;
close all;
figure()
warning off;
disp('**************************************************************')
disp('*****************Coding & Decoding Image 1******************')
disp('**************************************************************')
% Receive the coded Image from encoder1
[Xone MM gamma1] = encoder1(image1,rate3,h);
% Deocode to retrieve Image1
X1 = WaveDeQuant(Xone,MM,gamma1,5,'db1',sigma,0,h);
% For Testing Purposes Only
    X11 = readimage(image1);
    PSNR__Image1 = calcPSNR(X11,X1)
hold on; grid; xlabel('bpp'); ylabel('PSNR-Image2');
% The bit-rates that are used for Image2
rate = [0 0.2 0.3 0.4 0.5];
[aaa,bbb] = size(rate);
for j = 1:3
    if j==1
        k=0;
        str = '-.k*';

disp('***************************************************************************
******')
        disp('*************************Coding & Decoding Image
2*************************')
       str2=['*******************' 'Sigma = ' num2str(sigma) ', No
denoising algorithm' '*********************'];
        disp(str2);

disp('***************************************************************************
******')
        str2 = ['sigma = ' num2str(sigma) ' No denoising algorithm'];
        str11 = str2;
    end
    if j==2
        k=1;
        str = '-.bo';

disp('***************************************************************************
******')
        disp('*************************Coding & Decoding Image
2*************************')
       str2=['***************' 'Sigma = ' num2str(sigma) ', NormalShrink
denoising algorithm' '*****************'];
        disp(str2)

disp('***************************************************************************
******')
        str2 = ['sigma = ' num2str(sigma) ' NormalShrink denoising
algorithm'];
        str22 = str2;
    end
    if j==3
        k=0;
        sigma=0; %=>no_noise
        str = '-.r+';

disp('**********************************************************************
******')
        disp('************************Coding & Decoding Image
2*************************')
       str2=['*************************' 'No noise - Joint Coding'
'*************************'];
        disp(str2)

disp('**********************************************************************
******')
        str2 = [ 'No noise - Joint Coding' ];
        str33 = str2;
    end
for i = 1:1:bbb
    % Receive the residual image, motion vectors, the allocated bits and
    % the maxima and minima of image2
    [vert_disp,horz_disp,par,M_par,gamma2] =
encoder2(image2,sigma,Xone,MM,gamma1,rate(i),k,h);
    % Decode the residual image
    par = WaveDeQuant(par,M_par,gamma2,5,'db1',sigma,0,h);
    b = 8;  % block_size
    T = 512/b;
    % Estimate Image2 from Image1 using the motion vectors
    for m = 1:1:T
        for n = 1:1:T
            c = ((m-1)*b)+1;
            d = ((n-1)*b)+1;
            Q = c+b*vert_disp(m,n);
            W = d+b*horz_disp(m,n);
            X2_est(c:b*m,d:b*n) = X1(Q:Q+(b-1),W:W+(b-1));
        end
    end
    % Add the residual Image to estimated_Image2 for reconstruction of
    % Image2
    P=X2_est+par;
    % FOR TESTING PURPOSES ONLY
    X22 = readimage(image2);
    PSNR_Image2_DC(i,1) = calcPSNR(X22,P);
    PSNR__Image2_DC = calcPSNR(X22,P)
    % Rate that is used for Independent Coding of Image2
    rate2 = [0.1 0.2 0.3 0.4 0.5];

    % Independent Coding
    if j==3
        [TT M gamma3] = WaveQuant(X22,5,'db1',rate2(i),0,h);
        A = WaveDeQuant(TT,M,gamma3,5,'db1',sigma,0,h);
        PSNR_Image2_IC(i,1) = calcPSNR(X22,A);
        PSNR__Image2_IC = calcPSNR(X22,A)
    end
    bpp2_IC(i) = rate2(i);
    bpp2_DC(i) = rate(i)+(((2*T*T*3))/(512*512));
end
plot(bpp2_DC,PSNR_Image2_DC,str)
end
plot(bpp2_IC,PSNR_Image2_IC,'-.gd')

legend(str11,str22,str33,'Independent Coding');
```

```
% The codes below could be used for showing the Motion vector
%{
figure(1); hold on; axis([0 T 0 T]);
YY = meshgrid(T:-1:1)';
XX = meshgrid(1:1:T);
quiver(XX,YY,horz_disp,vert_disp)
%}
```

“N-Level Wavelet Transform”

```
function T = N_Level_DWT(A,N,filter);
% This function takes the N-th level wavelet transform of the given image
[a,b] = size(A);
T = zeros(a,a);
for i = 1:1:N
    [A,B,C,D] = dwt2(A,filter);
    [c,d] = (size(A));
    T(1:(c*2),1:(c*2)) = [A,B;C,D];
end
```

“N-Level Inverse Wavelet Transform”

```
function A = N_Level_IDWT(T,N,filter);
% This function calculates the inverse wavelet transform of the given
% transformed image
[a,b] = size(T);
cc = BSC(filter,N,a);
dd = [512 cc];
A = T((1:cc(N)),(1:cc(N)));
for i = 1:1:N
    c = cc(N-i+1);
    D = T((c+1):(2*c),(c+1):(2*c));
    C = T((c+1):(2*c),(1:c));
    B = T((1:c),((c+1):2*c));
    A = idwt2(A,B,C,D,filter,dd(N+1-i));
end
```

“Quantizer”

```
function [out,MIN,MAX] = Quantizer(B,gamma,h)
% This function quantizes the given image B, with the given number of bits
if h==1
    [a,b] = size(B);
    MIN = min(min(B));
    B = B - MIN;
    MAX = max(max(B));
    Q = 2^gamma;

    if Q==1
        out=zeros(a,b);
        MIN=0;
        MAX=0;
    else
        P = MAX/(Q); % The step-size
        if P > 0
            out = round(B/P);
        else
            out = zeros(a,b);
        end
    end
else
```

```
    out = B; MIN =0; MAX=0;
end
```

"Dequantizer"

```
function out = Dequantizer(In,MIN,MAX,gamma,h);
% This function dequantizes "In" with the given MIN, MAX and gamma values
if h==1
    [a,b] = size(In);
    Q = 2^gamma;
    if Q == 1
        out=zeros(a,b);
    else
        P = MAX/(Q);
        tmp = In*P;
        out = round(tmp + MIN);
    end
else
    out = In;
end
```

"Bit_Alloc"

```
function gamma = bitalloc(T,N,filter,rate,v);
% The bitalloc function allocates bits to different subbands. This function
% uses a predefined weighting for bit allocation.
x = (3*N)+1;
if rate > 0
    % The weightings
    w = ones(1,x); w(1) = 0.178; w(2) = 0.178; w(3) = 0.044;
    w(4) = 0.561; w(5) = 0.561; w(6) = 0.284; w(9) = 0.727;

    [a,b] = size(T);
    cc = BSC(filter,N,a);
    gamma = zeros(1,x);

    % initializing the distortion matrix
    Dist = zeros(1,x);
    for j = 1:(12*x)       % assuming 12 bits is the maximum number of bits
assigned to a block
        for i = 1:1:N
            c = cc(i);
            B = T((1:c),((c+1):2*c));
            [B1,MIN,MAX] = Quantizer(B,gamma(3*i - 2)+1,1);
            B1 = Dequantizer(B1,MIN,MAX,gamma(3*i - 2)+1,1);
            tmp = sum(sum(abs(B1-B)))/(c^2);
            Dist(3*i - 2) = w(3*i - 2)*tmp;

            C = T((c+1):(2*c),(1:c));
            [C1,MIN,MAX] = Quantizer(C,gamma(3*i - 1)+1,1);
            C1 = Dequantizer(C1,MIN,MAX,gamma(3*i - 1)+1,1);
            tmp = sum(sum(abs(C1-C)))/(c^2);
            Dist(3*i - 1) = w(3*i - 1)*tmp;

            D = T((c+1):(2*c),(c+1):(2*c));
            [D1,MIN,MAX] = Quantizer(D,gamma(3*i)+1,1);
            D1 = Dequantizer(D1,MIN,MAX,gamma(3*i)+1,1);
            tmp = sum(sum(abs(D1-D)))/(c^2);
            Dist(3*i) = w(3*i)*tmp;
        end
        A = T(1:c,1:c);
        [A1,MIN,MAX] = Quantizer(A,gamma(x)+1,1);
        A1 = Dequantizer(A1,MIN,MAX,gamma(x)+1,1);
        tmp = sum(sum(abs(A1-A)))/(c^2);
        Dist(x) = tmp;

        % At low bit rates setting B1,C1,D1,B2,C2 and D2 to zero would
        % generally help
        if v==1
            Dist(1:6) = 0;
        end

        % We need to find the subband that has the highest distortion as we
        % are starting from 0bits for all subbands. We also need to make
        % sure that the allocated bits of no subband goes above 12.
        [a,b] = sort(Dist,'descend');
        for k = 1:x
            if gamma(b(k)) >= 12
            else
                gamma(b(k)) = gamma(b(k)) + 1;
                break;
            end
        end
        % Find the total rate using CRC function
        R = CRC(gamma,cc);
        if R >= rate
            break;
        end
    end
else
    gamma = zeros(1,x);
end
```

“WaveQuant”

```
function [out M gamma] = WaveQuant(T,N,filter,rate,v,h)
% The function WaveQuant is a multi process function. It first takes the
% wavelet transform of a given image T. Then it allocates bits to different
% subbands at the given bit-rate, using the bitalloc function. Finally it
% quantizes all the subbands and outputs the transformed image. At the end
% the inverse wavelet transform of the image is calculated

% Taking the wavelet transform (for Haar use 'db1')
T = N_Level_DWT(T,N,filter);

% Allocating bits to different subbands at the given bit-rate
gamma = bitalloc(T,N,filter,rate,v);
[a,b] = size(T);

% Finding the sizes of the different subbands
cc = BSC(filter,N,a);
out = zeros(a,b);

% Finding the number of subbands
x = (3*N)+1;

% Initializing the Minima/Maxima matrix
M = zeros(2,x);
for i = 1:1:N
```

```
    c = cc(i);
    B = T((1:c),((c+1):2*c));
    [B,M(1,(3*i - 2)),M(2,(3*i - 2))] = Quantizer(B,gamma(3*i - 2),h);

    C = T((c+1):(2*c),(1:c));
    [C,M(1,(3*i - 1)),M(2,(3*i - 1))] = Quantizer(C,gamma(3*i - 1),h);

    D = T((c+1):(2*c),(c+1):(2*c));
    [D,M(1,(3*i)),M(2,(3*i))] = Quantizer(D,gamma(3*i),h);

    A = zeros(c,c);   % temporary
    if i == N
        A = T(1:c,1:c);
        [A,M(1,x),M(2,x)] = Quantizer(A,gamma(x),h);
    end

    out(1:(c*2),1:(c*2)) = [A,B;C,D];
end
```

"WaveDeQuant"

```
function out = WaveDeQuant(T,M,gamma,N,filter,sigma,k,h)
% The function WaveDeQuant is a multi process function. It first finds the
% inverse quantization of the wavelet subbands. Denoising will be applied
% on the dequantized subbands. k is a flag used for enabling/disabling the
% denoising algorithm.
[a,b] = size(T);
out = zeros(a,b);
cc = BSC(filter,N,a);
%var = calcvar(T,M,gamma,cc,h);
x = (3*N)+1;
for i = 1:1:N
    c = cc(i);

    B = T((1:c),((c+1):2*c));
    B = Dequantizer(B,M(1,(3*i - 2)),M(2,(3*i - 2)),gamma(3*i - 2),h);
    B = denoiseNS(B,N,sigma,k);

    C = T((c+1):(2*c),(1:c));
    C = Dequantizer(C,M(1,(3*i - 1)),M(2,(3*i - 1)),gamma(3*i - 1),h);
    C = denoiseNS(C,N,sigma,k);

    D = T((c+1):(2*c),(c+1):(2*c));
    D = Dequantizer(D,M(1,(3*i)),M(2,(3*i)),gamma(3*i),h);
    D = denoiseNS(D,N,sigma,k);

    A = zeros(c,c);   % temporary
    if i == N
        A = T(1:c,1:c);
        A = Dequantizer(A,M(1,x),M(2,x),gamma(x),h);
    end

    out(1:(c*2),1:(c*2)) = [A,B;C,D];
end
out = N_Level_IDWT(out,N,filter);
out = round(out);
```

"denoise_NS"

```
function T = denoiseNS(T,N,sigma,k)
% The denoiseNS function is the implementation of NormalShrink denoising
% algorithm
if k==1
    [a,b] = size(T);
    beta = sqrt(log10(a/N));
    tmp = reshape(T,[],1);
    sd = std(tmp);
    thr = beta*(sigma^2)/sd;
    for m = 1:1:a
        for n = 1:1:b
            T(m,n) = sign(T(m,n))*max(0,abs(T(m,n))-thr); %soft
thresholding
        end
    end
end
```

“Calc_Var” (just for reference)

```
function var = calcvar(T,M,q,cc,h)
% The calcvar function estimates the variance of noise. This function is
% not used when we are working at low-bit rates.
[a,b] = size(M);
tmp=0;
for i = 1:((b-1)/3)
    if (q(3*i) >= 1)
        DD =
Dequantizer(T((cc(i):2*cc(i)),(cc(i):2*cc(i))),M(1,3*i),M(2,3*i),q(3*i),h);
        S = reshape(DD,[],1);
        S = median(abs(S));
        var = (S/0.6745)^2;
        tmp=1;
        break;
    end
end
if tmp==0
    var=0;
end
```

“read_image”

```
function out = readimage(name)
% The function readimage reads a given image to MATLAB. An example would be
% readimage('abc.bmp'). If the image is RGB then it is converted to
% grayscale.

[out map] = imread(name);
if max(size(size(out))) == 3
    out = rgb2gray(out);
end
out = double(out);
```

“BSC”

```
function cc = BSC(filter,N,a)
% The BSC function finds the sizes of the different wavelet subbands. This
% is necessary when we are working with Daubechies filters

% Get the vanishing moment number from the "filter" string
tmp = sscanf(filter, '%c', [1 3]);
tmp = (str2num(tmp(3))-1)*2;
```

```
cc(1) = ceil(a/2);
for i = 2:N
    cc(i) = ceil((cc(i-1)+tmp)/2);
end
```

“CRC”

```
function out = CRC(gamma,cc);
% The CRC function calculates the total rate by multiplying each
% subband by its number of allocated bits and the end dividing the sum of
% all rates by 512*512. We are assuming that our images are 512x512 pixels
[y,x] = size(gamma);
out=0;
N = (x-1)/3;
for i=1:N
    c = cc(i);
    out = out + ((gamma(3*i - 2)+gamma(3*i - 1)+gamma(3*i))*(c^2));
end
out = out + (gamma(x)*c^2);
out = out/(512*512);
```

“Calc_PSNR”

```
function out = calcPSNR(a,b);
% The function calcPSNR calculates the PSNR of a with regards to b or vice
% versa
[cc,dd] = size(a);
J = (a-b).^2;
MSE = (sum(sum(J))/(cc*dd));
out = 20*log10(255/sqrt(MSE));
```

“Channel_Noise”

```
function X1 = channel_noise(sigma,X1);
% The channel_noise function adds noise to the image at the given standard
% deviation
noise = sigma*randn(512,512);
X1 = X1+noise;
```