

FLIC: Fast & Lossless
Image Compression
Algorithm for
Hyper-Spectral Images

By: Dan, Liu
Guangyu, Cheng
Hongbo, Jiang
Meng, Hu

1	Introduction.....	3
1.1	Image Data	3
1.1.1	Temporal Redundancy.....	4
1.1.2	Spectral Redundancy.....	4
1.1.3	Spatial Redundancy.....	5
1.2	Requirements	5
1.3	Existing Methods	5
1.3.1	Lossless Text Compression	5
1.3.2	Lossless Image Compression	6
2	FLIC.....	8
2.1	System Overview	8
2.2	Inter-Frame Encoder	8
2.3	Inner-Frame Encoder	9
2.3.1	Prediction Steps.....	10
2.3.2	Linear Regression.....	11
2.3.3	Computing Prediction Value	12
2.3.4	Compute Prediction Error	14
2.3.5	Post-processing	15
2.4	Entropy Encoder	16
2.4.1	Huffman Encoding	16
2.4.2	Arithmetic Coding:.....	16
3	Experiment Results	18
3.1	Experiment Environment	18
3.2	Results.....	18
3.2.1	Arithmetic Encoding	19
3.2.2	Huffman Encoding	20
3.2.3	Effect of Each Stage	20
4	Conclusion	22
5	References.....	23

1 Introduction

Image compression is the process of reducing the number of bits required to represent an image. Hyper-spectral images are a particular class of images that require specialized coding algorithms. Remotely sensed imagery, which is the image source in our project, is a common class of multi-spectral data. Remote sensing consists of capturing image data from a remote location. The sensing platform is usually an aircraft or satellite, and the scene being imaged is usually the earth's surface. Since the sensor and the target are so far apart, each pixel in the image can correspond to tens or even hundreds of square meters on the ground. In particular, a larger number of spectral bands are necessary if a single data set is to be used for multiple purposes. Compression is important for this class of images both to minimize transmission bandwidth from the sensing platform to a ground station and to archive the captured images.

Because hyper-spectral images are three-dimensional data sets, a straightforward extension of two-dimensional image compression algorithms is generally not appropriate. In addition, the hyper-spectral images in our domain have some special properties, for example, the data of two adjunct areas within an image may be highly correlated. Because of this kind of characteristic, we can achieve better performance by proposing a new algorithm for geographical hyper-spectral image compression.

In this project, we proposed and implemented a new algorithm: FLIC to compress a sequence of hyper-spectral images. Our algorithm consists of three phases: in the first stage, difference between two consecutive frames is computed; then a linear predictor is used to predict pixel value based on its several neighbors; finally, an entropy encoder is implemented as a back-end encoder. Each of these three steps handles the temporal redundancy, the spatial redundancy and the entropy redundancy respectively. This algorithm can achieve an average compression rate of 4.75:1 and is almost real-time.

In Section 1, first we will analyze the image source data and address the requirements of our project; then several existing image compression methods will be briefly explored to show why a new algorithm is needed; In Section 2, a new compression algorithm for hyper-spectral images (FLIC) will be proposed, and each steps of the algorithm will be explained in details; The experiment results will be presented and evaluated in Section 3. In Section 4, we will address some possible future improvements of FLIC and draw a conclusion.

1.1 Image Data

The source images are taken in a time sequence by a hyper-spectral sensor on an aircraft. The aircraft is flying from north to south over a region, until the whole region

is covered.

In a single flight, about 2500 images will be recorded. Each of them is called a scan line of the whole region. These images are in RAW format, which record the spectral image of each scan line. Each RAW file is formatted as 752 pixels by 240 bands. Every pixel value is represented in 16 bits, and the actual digitization is at 10-bit depth, which is stored in 10 less significant bits of the 16 bits. The images are captured at a scan rate of 20 scans per second.

The size of a single RAW file is about 350 kilobytes. For one flight, the size of 2500 images can be around 846 megabytes.

There are mainly three kinds of redundancy that can be exploited in hyper-spectral images, namely, temporal redundancy (inter-frame redundancy), spectral (inter-band) redundancy and spatial redundancy. For each of them, some algorithms can be adopted to remove the redundancy in order to achieve a compression. In addition, traditional text data compression algorithms can serve as the “back-end” process of the whole system.

Next, we will briefly explore these kinds of redundancy in hyper-spectral images.

1.1.1 Temporal Redundancy

Because of the moving manner of remote-sensing camera over the region, there will be some correlation between two consecutive images. Because the contents of two images are adjacent areas, the data of two frames will not differentiate too much.

Based on this observation, we can apply frame encoding on the input image sequence. The later image frame can be encoded based on the previous image frame, that is, only the difference between two consecutive images is recorded. However, this approach suffers from the following fact. If we lost some previous image data by accident, the following image frames can not be recovered. To relieve the problem, we can record a complete frame without any frame encoding between several frames. Because of the presence of these reliable frame data, the following compressed image is more likely to be recovered.

1.1.2 Spectral Redundancy

As stated earlier, hyper-spectral images are more complicated in that they are three-dimensional data sets.

Data in different bands can be correlated, which means that one band can be fully or partly predicted from other bands). However, the third (spectral) dimension is qualitatively different from the spatial dimensions, and it generally cannot be modeled as stationary. The correlation between adjacent spectral bands, for example, can vary widely depending on which spectral bands are being considered.

1.1.3 Spatial Redundancy

Within a single hyper-spectral image, the data of two adjunct areas may be highly correlated.

Pixel-to-pixel correlation is the form of spatial redundancy that is most often exploited. A normalized autocorrelation function for pixels in an image is defined in [1]. The behavior of the autocorrelation function near the origin reveals the degree of local redundancy present in the data. For example, DPCM exploits this redundancy by forming a prediction of the next pixel value based on past values, and coding the residual error of the prediction.

1.2 Requirements

The following are the specific requirements of our project.

A lossless compression algorithm is required. Because these images are for geographical use, information needs to be preserved as complete as possible. That is to say, the original image can be identically recovered from the compressed image.

A real-time compression algorithm is required. In the problem we studied, the compression is going to be done in real time during the image capturing. That is to say, once the sensor has an image ready, it is compressed, and then saved in the repository. Since the aircraft carrying the sensor is moving very fast and the frequency of taking images is high, the compression is expected to be done as fast as possible. Otherwise, the compression will be of no practical use.

1.3 Existing Methods

Image compression has been an active research area for more than 30 years. Existing methods can be divided into two categories, namely, lossless compression and lossy compression. In lossless compression, the original image can be reconstructed from a compressed image without any loss of information. On the other hand, lossy compression will eliminate some original image information permanently after compression. Many algorithms have been proposed for both lossless and lossy image compression, for example, PNG, JPEG and wavelet transformation.

First, we will look at some traditional lossless text compression methods, which are also suitable for image compression; then we will explore some lossless compression methods specifically designed for images.

1.3.1 Lossless Text Compression

Most of the text data compression methods in common use today fall into one of two categories: dictionary based schemes and statistical methods. In the world of

small systems, dictionary based data compression techniques are more popular. A well-known example of a dictionary technique is LZW data compression. However, by combining arithmetic coding with powerful modeling techniques, statistical methods for data compression can actually achieve better performance. Among them, Huffman Coding is the best known method. There are strict limits to the amount of compression that can be obtained with lossless compression. Lossless compression ratios are generally in the range of 2:1 to 8:1.

The following are some of the popular text data compression algorithms:

1) LZW

The Lempel-Ziv-Welch algorithm is a variable-to-fixed length code. The key of the method is that it automatically builds a dictionary of previously seen strings in the text being compressed. The compression process replaces strings of characters with single codes, which refer to entries in the dictionary. It is firstly presented by Abraham Lempel and Jacob Ziv in [2], then later improved by Terry Welch[3] in 1984.

LZW is the compression algorithm used in the GIF graphics file format, which is one of the standard graphic formats used on the World Wide Web.

2) Huffman coding

The basic idea in Huffman coding is to assign short code words to those input blocks with high probabilities and long code words to those with low probabilities. It is firstly published by D.A Huffman in [4].

Huffman coding today is often used as a "back-end" to some other compression method. DEFLATE (PKZIP's algorithm) and multimedia codecs such as JPEG and MP3 have a front-end model and quantization followed by Huffman coding.

3) Derivatives of Huffman Coding

Several adaptive coding techniques have been proposed as a derivative or generalization of Huffman Coding, including dynamic Huffman coding [5], adaptive arithmetic coding [6] and the Rice algorithm [7].

1.3.2 Lossless Image Compression

Historically, lossless image compression inherited the theoretical framework of text data compression. Undoubtedly, all the above compression algorithms can be directly used in image compression. However, they are initially devised for text file compression, and do not consider the characteristics of image data. Therefore, it is possible to achieve better compression performance by proposing new algorithms specifically for image data.

Among all existing lossless image compression methods, context based predictive techniques are considered to be the most efficient one. Two basic parts of predictive compression scheme are prediction of the current pixel from the information of previously encoded pixels (contexts for prediction) and coding of the prediction errors by using entropy coders such as a Huffman coder or an arithmetic coder.

There are many algorithms proposed for lossless image compression over years, below are several typical methods:

1) Rice Machine

Rice machine is an early proposed lossless image compression algorithm [8], which employs DPCM (Differential Pulse Code Modulation). The prediction errors were encoded in blocks of, say, 16 pixels.

2) General Linear/Non-Linear Prediction

Linear predictor is more complex than DPCM. A linear predictor is a weighted sum of local pixel values, where the weights sum to unity. In general, if the weights are positive and sum to unity there is no "increase" in the noise (hence accuracy) of the predictor. Nonlinear predictors have also been successfully used. The nonlinear predictors that are most popular use a median value.

3) Context-Dependent Compression

In [9], the proposed approach uses quantized prediction errors from local pixel location already encoded as the context. With the context-dependent treatment of the prediction error distributions, it can offer higher compression ratio.

Many other context-based compression algorithms are presented after this, including JPEG-lossless, Sunset, etc.

Due to the spatial and temporal characteristics of image data, different context-based models and predictive models can be applied to achieve better compression performance. In addition, the hyper-spectral images in our domain have some special properties. Existing text and image compression algorithms have their own properties, but may not work efficiently in our problem domain. When faced with a specific problem, we can often come up with a simpler more efficient solution. Therefore, it is practical and important for us to propose a new algorithm for geographical hyper-spectral image compression.

2 FLIC

In this section, a fast and lossless image compression algorithm (FLIC) for hyper-spectral images is proposed. Each step of the algorithm is explained in details, and some of the implementation issues are discussed.

2.1 System Overview

Basically, our algorithm is divided into three stages. The first stage is the inter-frame encoder, in which difference between two consecutive frames is computed; the second stage is the inner-frame encoder, which deals with the spectral and spatial redundancy within the difference frame; an entropy encoder follows the inner-frame encoder. The output of inner-frame, which is the prediction error values of a linear predictor, will be encoded by the entropy encoder using Arithmetic encoder or Huffman encoder. Fig 2.1 shows the building blocks of our algorithm.

The decoding process is the reverse of the encoding process.

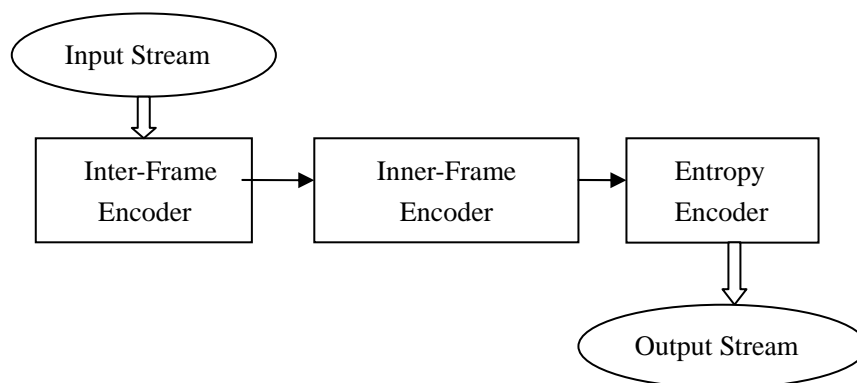


Fig 2.1 Encoding Process

In the following subsections, each stage of our system will be explained.

2.2 Inter-Frame Encoder

Because the contents of two consecutive images are adjacent areas, there exists some similarity between image data of two frames.

We test the image data by choose the pixel value at a fixed location (100, 100) from the sequence of all frames. Fig 2.2 shows that the correlation between consequence two frames is very high. The pixel values in two consecutive frames seldom change dramatically, which indicates that the difference between these two

values is rather small at most time.

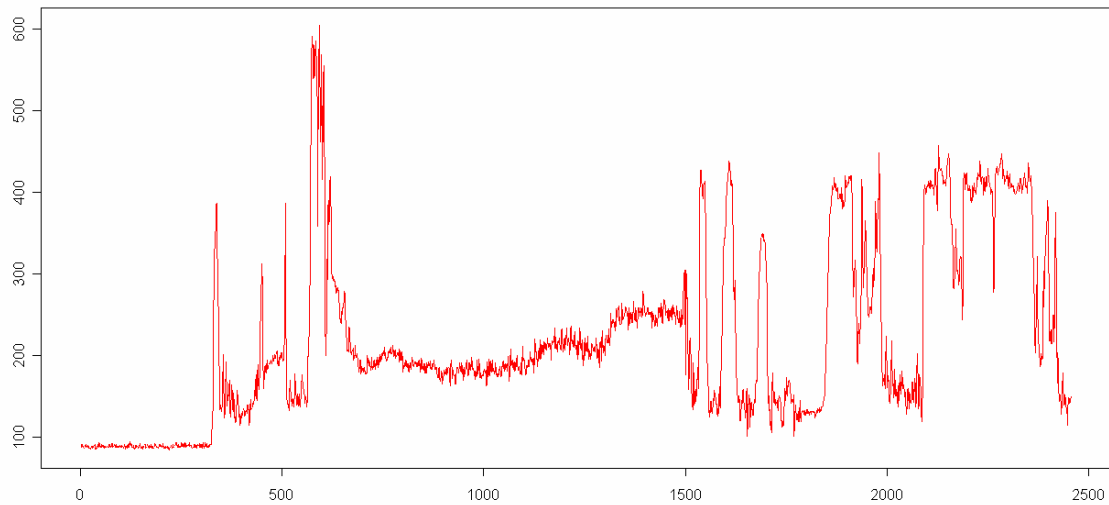


Fig 2.2 Value of pixel (100,100) in all frames

A very importance property of information redundancy compression is that there is many same values in the sequence after some transform operation. Therefore, we set up a simple experiment to test the data distribution. Figure 2.3 shows the result of data distribution before and after difference operation. That is, after the difference operation between two frames, if we can exploit appropriate coding algorithm for the result, the redundancy information should be compressed.

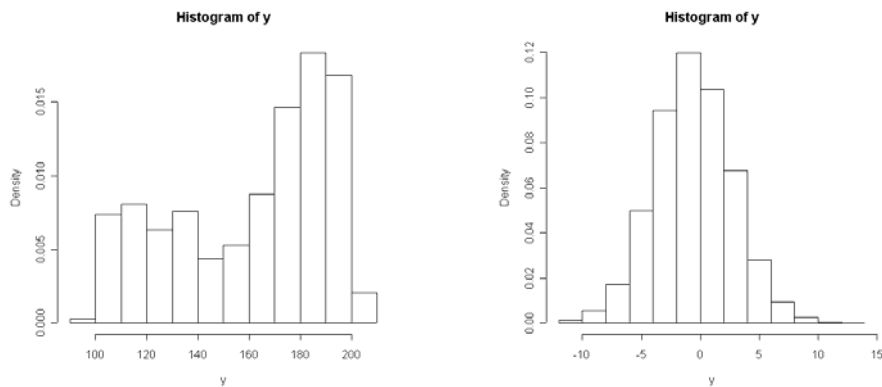


Fig 2.3 Distribution before and after difference operation

Based on the above observation, in our project, we exploit a difference operation between every two consecutive frames. The difference values are recorded as the input of next step.

2.3 Inner-Frame Encoder

Among the remaining two stages of the lossless compression of hyper-spectral images (prediction, and entropy-coding), prediction stage is performed followed by entropy coding of the prediction errors, and typically act as a fairly important role in

reducing the complexity of the probability modeling done by the entropy coder, hence reducing the modeling cost of coding. The most common used predictors are linear predictor where all the pixels used to make the prediction have the related spatial location to the current pixel, and special coefficients are used for optimize, i.e. to minimize the error inside the prediction of each node.

Linear prediction has been highly used in the compression of various image standards. For example, JPEG has a lossless variant, which uses linear prediction for data compression. During the linear prediction, the values of individual pixels are predicted based on the values of surrounding pixels. Rather than coding each individual pixel value, the difference between the prediction and reality is coded according to certain linear prediction function. The lossless JPEG standard is not used very much because the compression achieved is much less than that achieved by lossy JPEG.

Another significant example of prediction in image compression is video compression, which can be thought of as repeated image compression over a period of time, since video is a series of images. Most video compression methods use the first frame to predict the second frame, the second frame to predict the third frame, etc. The difference between the predicted frame and the actual frame is encoded. To view the video, the receiver or playback mechanism uses the prediction, the prediction scheme and the previously reconstructed frame to show the next frame. Motion compensation, predicting the motion of objects during a series of frames, is also a factor in video compression, since more objects moving, or objects moving quickly, increases the amount of data to be compressed. Understandably, video compression is very complicated and the end result is still a lot of data.

In our project, the remote sensed hyper-spectral images show two forms of correlation: spatial (the same material tends to be present in many adjacent pixels) and spectral (one band can be fully or partly predicted from other bands). The spectral correlation is generally much stronger than the spatial correlation. Furthermore, dynamic range and noise levels (instrument noise, reflection interference, aircraft movements, etc.) of the image data are higher than those in photographic images. For these reasons, we need to make full use of both spatial and spectral prediction to transform the original image data to prediction error data which will be much more suitable for entropy coding.

Motivated by these considerations, we propose a new approach that uses a new linear predictor for handling both inter-band correlation and spatial correlation in neighbor pixels.

2.3.1 Prediction Steps

There are actually two steps in the inner-frame encoder: linear regression and prediction. The aim of linear regression is to calculate the best coefficients for prediction function, instead of using fixed ones. Once we have the coefficients, the prediction is done and the prediction error is to be recorded.

2.3.2 Linear Regression

A classic statistical problem is to try to determine the relationship between two random variables X and Y . For example, we might consider height and weight of a sample of adults. Linear regression attempts to explain this relationship with a straight line fit to the data. The linear regression model postulates that

$$Y = a + bX + e$$

where the "residual" e is a random variable with mean zero. The coefficients a and b are determined by the condition that the sum of the square residuals is as small as possible.

In our project, we adopt multiple linear regressions. Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y . The population regression line for p explanatory variables x_1, x_2, \dots, x_p is defined to

be $\mu_y = \beta_0 + \beta_{1 \times 1} + \beta_{2 \times 2} + \dots + \beta_{p \times p}$. This line describes how the mean response μ_y changes with the explanatory variables. The observed values for y vary about their means μ_y and are assumed to have the same standard deviation σ . The fitted values b_1, b_2, \dots, b_p estimate the parameters $\beta_0, \beta_1, \dots, \beta_p$ of the population regression line.

Since the observed values for y vary about their means μ_y , the multiple regression model includes a term for this variation. In words, the model is expressed as DATA = FIT + RESIDUAL, where the "FIT" term represents the expression $\beta_0 + \beta_{1 \times 1} + \beta_{2 \times 2} + \dots + \beta_{p \times p}$. The "RESIDUAL" term represents the deviations of the observed values y from their means μ_y , which are normally distributed with mean 0 and variance σ . The notation for the model deviations is ε .

Formally, the model for multiple linear regressions, given n observations, is

$$y_i = \beta_0 + \beta_{1 \times i1} + \beta_{2 \times i2} + \dots + \beta_{p \times ip} + \varepsilon_i, \text{ for } i = 1, 2, \dots, n.$$

In the least-squares model, the best-fitting line for the observed data is calculated by minimizing the sum of the squares of the vertical deviations from each data point to the line (if a point lies on the fitted line exactly, then its vertical deviation is 0). Because the deviations are first squared, then summed, there are no cancellations between positive and negative values. The least-squares estimates b_1, b_2, \dots, b_p are

usually computed by statistical software.

The values fit by the equation $b_0 + b_{1 \times i_1} + \dots + b_{p \times i_p}$ are denoted \hat{y}_i , and the residuals e_i are equal to $y_i - \hat{y}_i$, the difference between the observed and fitted values. The sum of the residuals is equal to zero.

The variance σ^2 may be estimated by $s^2 = \frac{\sum e_i^2}{n - p - 1}$, also known as the mean-squared error (or MSE). The estimate of the standard error s is the square root of the MSE. In our implementation, we use 4 variables linear regressions, and then there are 5 coefficients should be computed finally. That is, a_0, a_1, a_2, a_3, a_4 in equations below:

$$\begin{cases} L_{11}a_1 + L_{12}a_2 + L_{13}a_3 + L_{14}a_4 = L_{10} \\ L_{21}a_1 + L_{22}a_2 + L_{23}a_3 + L_{24}a_4 = L_{20} \\ L_{31}a_1 + L_{32}a_2 + L_{33}a_3 + L_{34}a_4 = L_{30} \\ L_{41}a_1 + L_{42}a_2 + L_{43}a_3 + L_{44}a_4 = L_{40} \end{cases} \quad \begin{aligned} L_{jk} &= \sum_{i=1}^n (x_{ji} - \bar{x}_j)(x_{ki} - \bar{x}_k) \\ L_{j0} &= \sum_{i=1}^n (x_{ji} - \bar{x}_j)(y_i - \bar{y}) \end{aligned}$$

$$a_0 = \bar{y} - a_1\bar{x}_1 - a_2\bar{x}_2 - a_3\bar{x}_3 - a_4\bar{x}_4 \quad 1 \leq j, k \leq n$$

2.3.3 Computing Prediction Value

Our linear predictor is based on the prediction coefficients resulting from the linear regression and consists of two main steps: pixel prediction and post-processing of the prediction results. Having obtained a linear prediction function with five coefficients through the previous Multiple Linear Regression step, the remaining task is to compute the prediction value of each pixel and then its prediction error. In the decompression processing, we still need to compute the prediction value of each pixel in order to recall the true value by using the error value to correct prediction value. The whole processing is shown in Fig 2.4 (compression) and Fig 2.5(decompression).

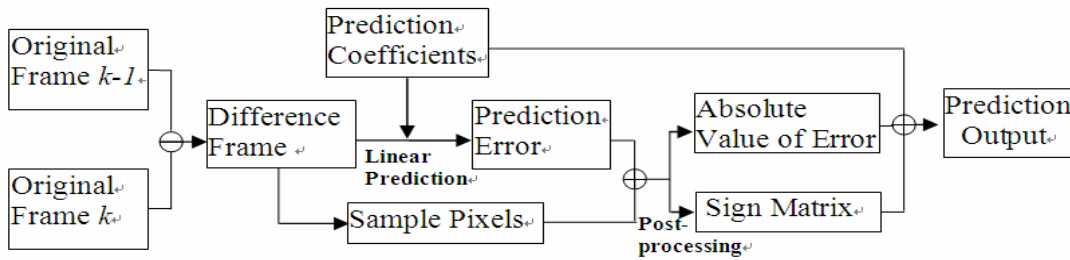


Fig 2.4 Detailed Compressing Process

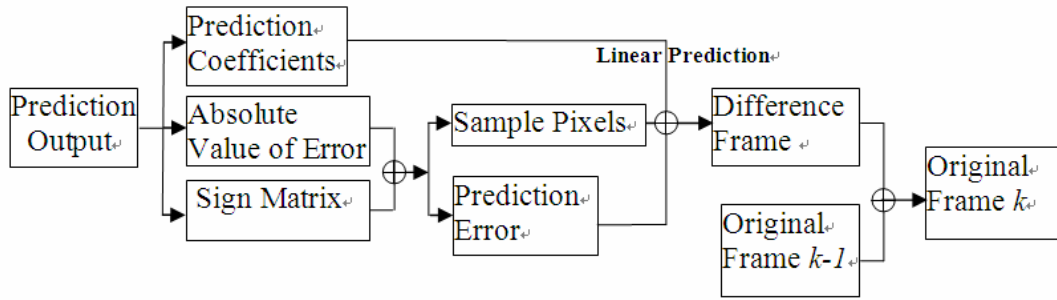


Fig 2.5 Detailed Decompressing Process

Our prediction scheme is described in Fig 2.6. For a given pixel (i, j) on the i th row and j th column, the predicted pixel value $\hat{p}(i, j)$ is computed according to a small surrounding range of the current pixel. Such a range is chosen such that it only consists of those pixels whose values are already known prior to the coding of the current pixel. In our algorithm, for those pixels not in the last column, we choose the three previous pixels above the current pixel and the neighbor pixel just ahead of the current pixel in the same row. Such a range is obviously not suitable for those pixels in the last column, thus we just use the two previous pixels above the current pixel together with the front neighbor pixel as the reference of our prediction computation.

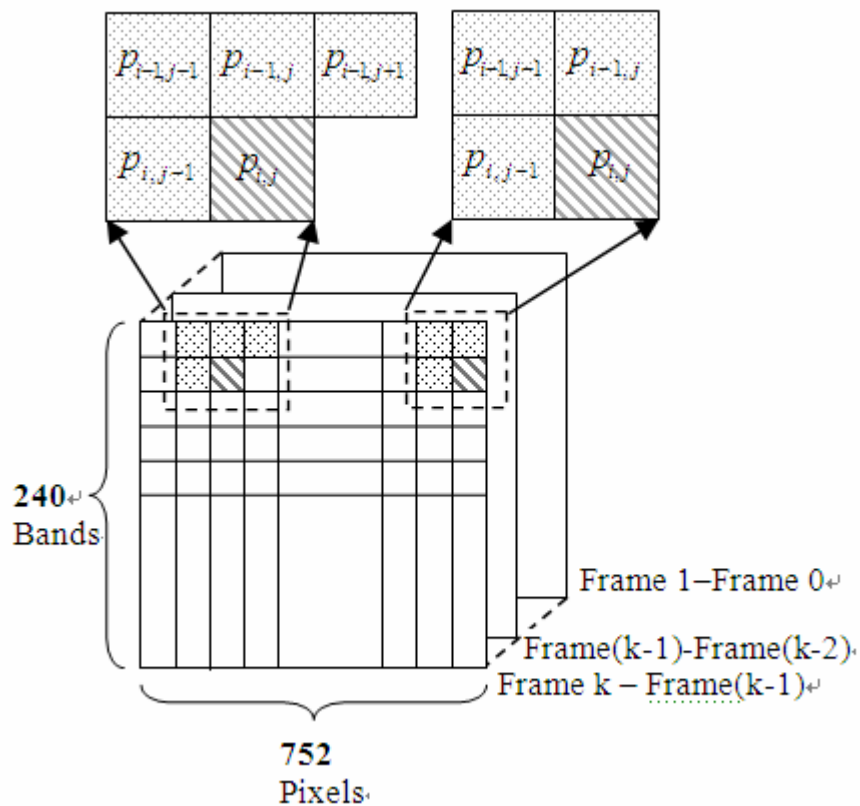


Fig 2.6 Prediction Scheme

Then, for each objective difference frame, we select the first row and the first column as the reference sample pixels, and use the following prediction formula to predict the rest part of the frame.

$$\hat{p}(i, j) = \begin{cases} a_0 + a_1 p(i, j-1) + a_2 p(i-1, j-1) + a_3 p(i-1, j) + a_4 p(i, j+1) & j < \text{number of pixels} \\ a_0 + a_1 p(i, j-1) + a_2 p(i-1, j-1) + a_3 p(i-1, j) & j = \text{number of pixels} \end{cases}$$

In order to learn a suitable coefficients a_0, a_1, a_2, a_3 and a_4 for our formula we use the Linear Regression mentioned in the previous processing. In fact, the more sample pixels we take for prediction, the more it is close to the real value. However, since it will be more complex to implement and cost more time if adapting more number of coefficients in our prediction, we just choose a trade off number, i.e., four neighbors for our prediction.

In general, the set of predictor coefficients may be fixed for all images (global prediction), or may be varied from image to image (local prediction), or may even vary within an image to accommodate the local changes in image statistics (adaptive prediction). If an image is characterized as a stationary random field, a set of local predictor coefficients can be derived for each image that minimizes the mean squared prediction error. In our case, for the purpose of the real-time compression, we do not implement a frame-adaptive set of coefficients, but assume that the coefficients do not verify much from frame to frame. Although it may bring larger range of error values if using the same set of coefficients for all frames, the speed is improved by doing the Linear Regression just once at the beginning of the whole processing. Moreover, we plan to improve our method by computing the coefficients after every certain number of frames are obtained, which is obviously a trade off between two extreme strategies.

Since the prediction is started from some known sample pixel values, we take the original pixel values of the first row and the first line of the original difference frame as the initial input of the prediction function. That means, for each difference frame, we just need to record the first row and the first line but not the whole frame of pixels. Since the error will be coded by entropy coding algorithm, the total information need to record after all these steps is obviously much less than the original values. As we will point out in the later chapter, the compression ratio is over four.

As shown before, the predictor achieves good performance by adapting the four directional predictors and the Multiple Linear Regression based on information obtained from the sample pixels of the first original frame. The linear prediction function should perform well in smooth areas of images.

2.3.4 Computing Prediction Error

After computing the prediction value, the prediction error of our predictor is straight forwardly represented by the difference between the original and the prediction values, i.e., the error of the prediction is given by

$$pred_err(i, j) = p(i, j) - \hat{p}(i, j)$$

Where $p(i, j)$ is the original value of current pixel and $\hat{p}(i, j)$ is the prediction value obtained after the previous step. Since for every (i, j) where $i = 0$ or $j = 0$, there is $p(i, j) = \hat{p}(i, j)$, then accordingly for those (i, j) there is $pred_err(i, j) = 0$.

Obviously, the difference value will either be positive or negative, which may lower the efficiency of the coding of the prediction error for its wide range of domain. To obtain a higher efficiency, we take another strategy, i.e. extract the sign of the error to form a particular sign matrix. Since there will be only 1 or 0 in such sign matrix, it can be compressed to as high as 8 times less as the original size by using just 1 bit to represent every item (1 byte). As for the processed error value, which are the absolute values of the original error, would be more suitable for the next stage, entropy coding processing, because all the values turned into positive integers such that the domain is at least as half as the original.

2.3.5 Post-processing

It should be emphasize that, the prediction error is not the only information needed for the coder of construction or the decoder for the reconstruction of the input image, since the sample values, sign matrix as well as the five coefficients need to be stored too. So we should do some post-processing to well organize the prediction result.

As for the sample values, we can merge them into the error matrix. As we mentioned before, the first row and first line of the original pixels need to be record as sample pixels. Since the first row and first line of the error are all 0's, we can use these entries to store those sample values, which will make full use of the space of the prediction output.

As for the signal matrix, we attach it behind the error matrix within a place of $752*240/8$ bytes.

As for the prediction coefficients a_0, a_1, a_2, a_3 and a_4 , since when decoding the error matrix, we need to use these five coefficients to recall the original image, we need a method to store these coefficients for conveniently getting these coefficients which are of double type from a file for using them in decoding. The detail to store these coefficients is following:

(a) Once obtaining these five coefficients after the linear regression, we should retype them, i.e., cut off the last several digits behind point.

For example, the value 0.345634577 of a_0 changes to 0.3456345 after retyping, so as to other coefficients. Such cutting operation will not affect the efficiency of the prediction since the cut part is small enough.

(b) After predicting, these five coefficients are recorded at the end of the prediction output file. First the double type coefficient is enlarged by 10000000 times (e.g., 3456345 for the previous example). Then the result will be store within four bytes. Since when accessing a number from a file by use of the function `fgetc()`, only one byte but not four bytes can be read at on time, so they have to be stored one byte

by one byte by four times.

For example, we do $a0 \gg 24$ (shift 24 bits to the right) & $0x000000FF$ to get the first byte then store the byte into the result file, and then do $a0 \gg 16$ (shift 16 bits to the right) & $0x000000FF$ to get the second byte, etc.

It is easy to access these coefficients from the file by reading the according four bytes of a coefficient one by one and reconstructing it. Then after dividing the result by 10000000 we can recall the coefficient.

Finally we combined all these information into one output file as the input to the next stage, entropy encoding.

2.4 Entropy Encoder

Entropy encoding is the last phase of our algorithm. In this stage, the output of linear prediction, which is the prediction error, is encoded by an entropy encoder. Two different lossless entropy encoders are implemented in our project. One is Huffman encoder, and the other one is Arithmetic encoder. These two encoders are independent modules and can be plugged into the previous modules.

2.4.1 Huffman Encoding

David Huffman in 1954 designed a lossless coding procedure for a frame of values that has the smallest possible average code length. It assigns a single bit to the most probable value and successively more bits to less probable values. The weighted sum of (number of bits for a particular value * probability of that value's occurrence in the frame) yields the average bit size per value needed to transmit the frame. This number is less than or equal to the number of bits if each value was equally probable.

Due to the limitation of space, we do not elaborate on the details of Huffman encoding. Here in our project, a C++ implementation of Huffman en-decoder is used.

2.4.2 Arithmetic Coding

The output from an arithmetic coding process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to create the exact stream of symbols that went into its construction. In order to construct the output number, the symbols being encoded have to have a set probabilities assigned to them.

Once the character probabilities are known, the individual symbols need to be assigned a range along a "probability line", which is nominally 0 to 1. It doesn't matter which characters are assigned which segment of the range, as long as it is done in the same manner by both the encoder and the decoder.

Each character is assigned the portion of the 0-1 range that corresponds to its

probability of appearance. Note also that the character "owns" everything up to, but not including the higher number. So the letter 'T' in fact has the range 0.90 - 0.9999....

The most significant portion of an arithmetic coded message belongs to the first symbol to be encoded. After the first character is encoded, we know that our range for our output number is now bounded by the low number and the high number.

The algorithm to accomplish this for a message of any length is shown below:

```
Set low to 0.0
Set high to 1.0
While there are still input symbols do
    get an input symbol
    code_range = high - low.
    high = low + range*high_range(symbol)
    low = low + range*low_range(symbol)
End of While
Output low
```

The algorithm for decoding the incoming number works like this:

```
Get encoded number
Do
    Find symbol whose range straddles the encoded number
    Output the symbol
    Range = symbol low value - symbol high value
    Subtract symbol low value from encoded number
    Divide encoded number by range
Until no more symbols
```

3 Experiment Results

3.1 Experiment Environment

The algorithm is implemented in Visual C++ 6.0. All experiments run under a PC with Pentium IV 2.0G CPU, 512M RAM and 7200 rpm harddrive.

We use the image files in \12-23-29 Line2\Hyperspectral as the test dataset. There are totally 2458 RAW files under this directory, which are 2458 scan lines of a region.

A demo system is developed to visualize the performance of the compression.

Fig 3.1 is a screenshot of our demo system:

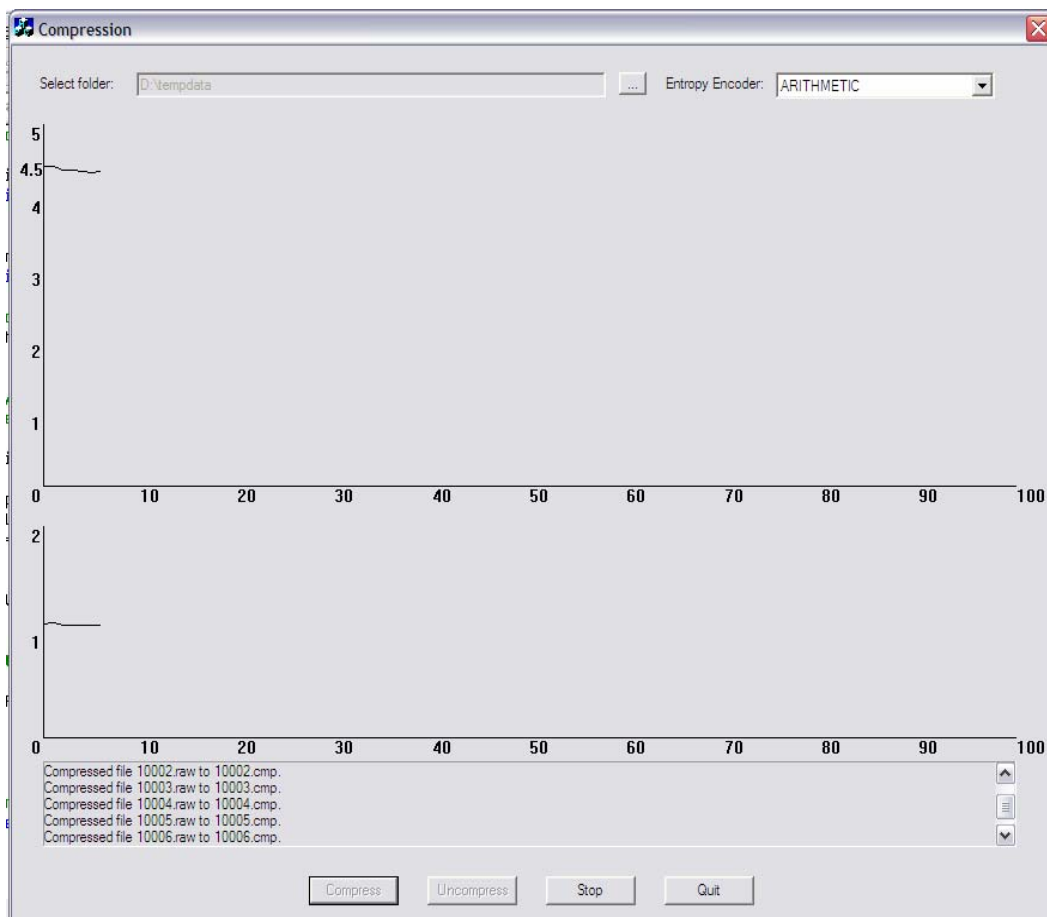


Fig 3.1 Demo Program

3.2 Results

We can evaluate our algorithm by the following measurements:

1. Average Compression Ratio;

2. Average Compression Time.

The 2458 RAW files are compressed by our system and the average compression ratio and average compression time are computed.

Then we compared the compression ratio and compression time of using two entropy encoders. We also remove each stage of our algorithm and compared the compression ratio by removing any stage. The results show that each stage in our algorithm contributes to the final compression ratio.

3.2.1 Arithmetic Encoding

Using Arithmetic Encoding as our back-end entropy encoder, we can achieve below performance:

Average Compression Ratio: 4.58: 1

Average Compression Time: 1.21 s

Fig 3.2 shows the compression ratio for the first 100 images. We can see that the system achieve a stable compression ratio of 4.5: 1.

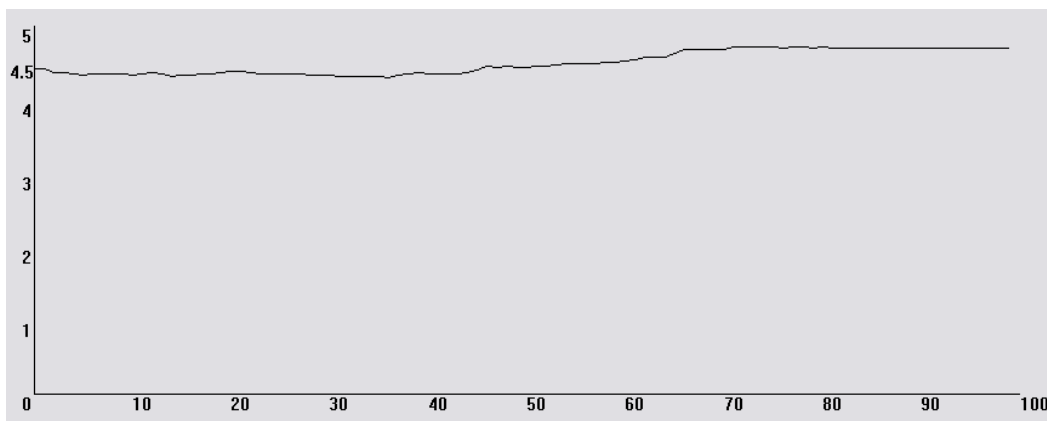


Fig 3.2 Compression Ratio of Arithmetic Encoding

Fig 3.3 shows the compression time for the first 100 images. The time for compressing one image is around 1.2 second.

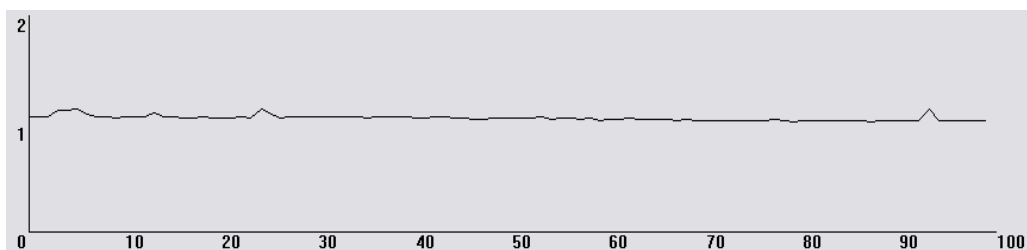


Fig 3.3 Compression Time of Arithmetic Encoding

3.2.2 Huffman Encoding

Using Huffman Encoding as our back-end entropy encoder, we can achieve below performance:

Average Compression Ratio: 3.05: 1

Average Compression Time: 0.24 s

Fig 3.4 shows the compression ratio for the first 100 images. We can see that the system achieve a stable compression ratio of 3: 1.

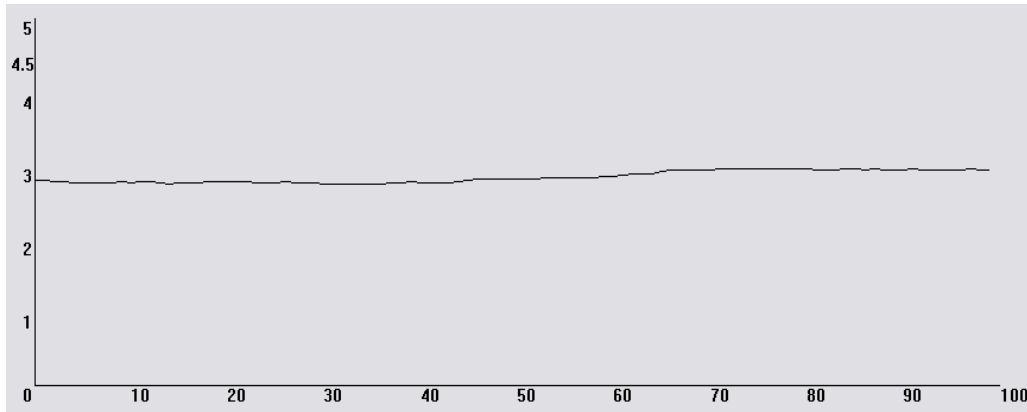


Fig 3.4 Compression Ratio of Huffman Encoding

Fig 3.5 shows the compression time for the first 100 images. The time for compressing one image is around 0.24 second.

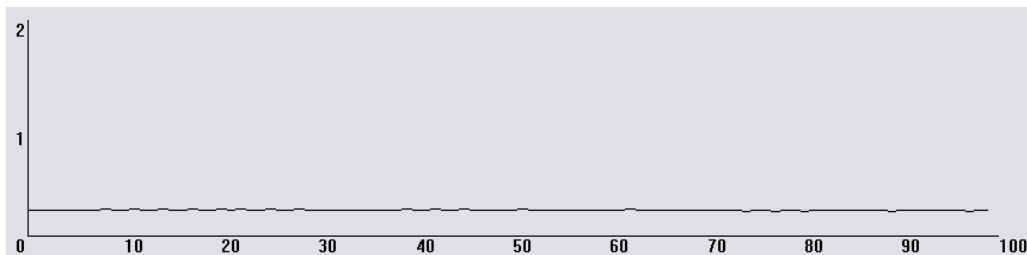


Fig 3.5 Compression Time of Huffman Encoding

3.2.3 Effect of Each Stage

In this experiment, we try to remove each stage of our algorithm, and get the compression ratio for just using the remaining steps to compress the image files.

For example, we may remove the frame difference computation step from the whole process, and directly use linear predictor on the original image file, then compress the prediction error with entropy encoder.

We tried to remove the following steps:

1. Difference computation between two consecutive frames.
2. Linear prediction based on neighbors.

3. Regression for linear predictor coefficients (Use fixed coefficients).
4. Extraction of signs in prediction error matrix.

Fig 4.6 shows the compression ratio for removing each of above steps, and the compression ratio of combining all these steps.

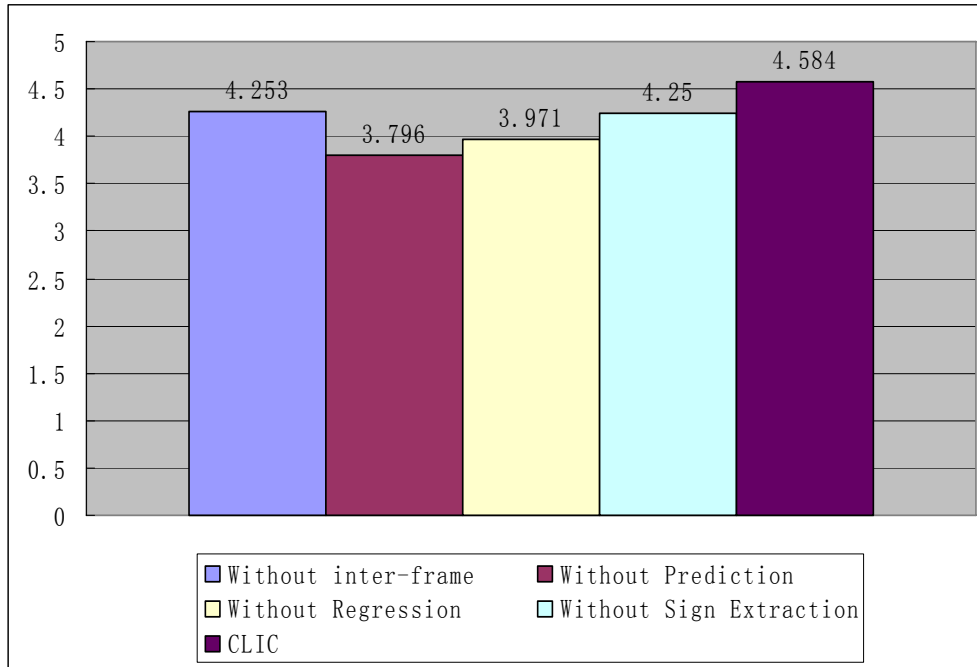


Fig 3.6 Compression Ratio Comparison of Removing Each Step

We can see that removing any step from the process will cause a decrease in compression ratio. Combing all the steps, which is represented in the purple column in Fig 4.6, will achieve the best compression ratio. This fact suggests that each stage of our algorithm contributes to the final compression ratio in some way.

A special fact need to be pointed out is that linear prediction plays a very important role in compressing the image data. From Fig 3.6, we can see that without prediction, the system has the lowest compression ratio.

4 Conclusion

In this project, a lossless compression algorithm for hyper-spectral image is proposed and implemented. The algorithm consists of three stages: inter-frame encoding, linear prediction and entropy encoding. In the first stage, the difference between two consecutive frames is computed; then a linear regression is conducted to calculate coefficients for linear prediction; based on these coefficients, a linear predictor predicts pixel values by several spatial and spectral neighbors, and the prediction error is recorded for entropy encoding; in the last stage, a Huffman encoder or an Arithmetic encoder is used to compress the prediction errors.

The experiment results show that our algorithm can achieve a compression ratio of 4.78: 1 in 1.2 s for a single image on a PC, when using the Arithmetic encoder for entropy coding. Compared to Arithmetic encoder, using Huffman encoder for entropy coding can be faster, but results in a smaller compression ratio.

However, there are some possible improvements can be done in the future:

1. Extern the 2-d LP to 3-d linear predictor instead of taking difference operation;
2. Use non-linear predictor, such as Quadric predictor;
3. Clustering band to choose a best reference band for predicting;
4. Use an adaptive Huffman encoder to fast the compressing process.

5 References

- [1] V. D. Vaughn and T. S. Wilkinson. System Considerations for Multispectral Image Compression Designs, *IEEE Signal Processing Magazine*, vol. 12, no. 1, pp. 19-31, January 1995.
- [2] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression, *IEEE Transactions on Information Theory*, Vol. 23, pp. 337--342, 1977.
- [3] T. A. Welch. A Technique for High-Performance Data Compression, *Computer*, pp. 8--18, 1984.
- [4] D.A. Huffman. A method for the construction of minimum redundancy codes. *Proc. IRE*, 40:1098-1101, 1951.
- [5] R.G. Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, IT-24(6):668674, November 1978.
- [6] T.M. Cover and J.A. Thomas. Elements of Information Theory. *Wiley Series in Telecommunications*. John Wiley and Sons Inc., 1991.
- [7] R.F. Rice. Some practical universal noiseless coding techniques. *Technical Report JPL Publication 79-22*, NASA, 1979.
- [8] R.F. Rice and J.R. Plaunt. Adaptive Variable-length Coding or Efficient Compression of Spacecraft Television Data, *IEEE Trans Commun.*, vol COM-19, no 6, December 1971, 889-897.
- [9] G. G. Langdon, Jr, J. Rissanen, and S. Todd, On Lossless Compression of Gray-scale Images, *Declassified proceedings paper of IBM Internal Conference on Data Compression of March 1980*.
- [10] J. Mielikainen, P. Toivanen, and A. Kaarna, "Linear prediction in lossless compression of hyperspectral images," *Opt. Eng.*, vol. 42, no. 4, pp. 1013–1017, 2003.
- [11] B. Aiazzi, P. S. Alba, L. Alparone, and S. Baronti, "Reversible compression of hyper-spectral imagery through inter-band fuzzy prediction and context coding," *IEEE Intl. Geosci. Remote Sensing Symp. '98*, pp. 2685–2687 ~1998!.
- [12] G. P. Abousleman. "Compression of Hyperspectral Imagery Using Hybrid DPCM/DCT and Entropy Constrained Trellis Coded Quantization," *Proceedings Data Compression Conference, IEEE*, Computer Society Press, 1995, 322-331.
- [13] J. Mielikäinen, A. Kaarna, and P. Toivanen. "Lossless Hyperspectral Image Compression via Linear Prediction," *Proceedings SPIE*, 2002, 4725:8, 600-608.
- [14] S. Subramanian, N. Gat, A. Ratcliff and M. Eismann. "Real-Time Hyperspectral Data Compression Using Principal Component Transform." *Proceedings AVIRIS Airborne Geoscience Workshop*, 1992.
- [15] M. J. Ryan and J. F. Arnold. "The Lossless Compression of AVIRIS Images By Vector Quantization," *IEEE Transactions on Geosciences and Remote Sensing* 35:3 (May), 1997,546-550.
- [16] G. P. Abousleman, T. T. Lam, and L. J. Karam. "Robust Hyperspectral Image

- Coding with Channel-Optimized Trellis-Coded Quantization," *IEEE Transaction on Geosciences and Remote Sensing* 40:4 , 2002,(April), 820-830.
- [17] G. Motta, F. Rizzo, and J. A. Storer. "Compression of Hyperspectral Imagery," *Proceedings Data Compression Conference, IEEE Computer Society Press*, 2003, 333-342.
- [18] Douglas C. Montgomery, Elizabeth A. Peck, G. Geoffrey Vining *Introduction to Linear Regression Analysis*, 3rd Edition. Wiley-Interscience. 2001
- [19] Douglas C. Montgomery, George C. Runger *Applied Statistics and Probability for Engineers*. John Wiley & Sons. 1998.
- [20] R. Snee. *Experiments in Industry: Design, Analysis, and Interpretation of Results (Chemical and Process Industries Division Technical Supplement)* Amer Society for Quality. 1985.