End-to-End System Consideration of the Galileo Image Compression System

K.-M. Cheung, M. Belongie, and K. Tong
Communications Systems and Research Section

In this article, we describe the image compression scheme of the Galileo spacecraft and the downstream data domain signal processing algorithms that enable efficient and robust transmission of data to Earth.

I. Introduction

We present a robust implementation of the Galileo image compression system. Due to the loss of the high-gain antenna, the mission was replanned to incorporate advanced data compression, error correction, and other downlink communication functions to maximize data return. Most instrument data will be highly edited and compressed before downlink through a noisy and unpredictable channel. The data compression schemes share a common disadvantage: channel errors in the compressed data tend to propagate in the reconstructed data. To ensure reliable communication, we incorporated a number of data domain error-recovery schemes in the downlink communication processing system. We developed a feedback concatenated decoder (FCD) for error correction and error detection and a containment strategy to prevent error propagation and realign the reconstructed data after decoder failure. We also developed a frame repair scheme that combats phase inversions and symbol insertion/deletion in the channel and a frame merging scheme that prevents data loss due to station hand over. The schematic diagram of the spacecraft and ground data processing systems are shown in Figs. 1 and 2, respectively.[1] The sample-domain gap processing scheme in the Deep Space Communications Complex Galileo telemetry subsystem (DGT) is not discussed here.

II. Image Compression

The Galileo image compression scheme is a block-based lossy image compression algorithm that uses an 8×8 integer cosine transform (ICT) [2]. The algorithm is similar in functionality to the baseline Joint Photographic Experts Group (JPEG) standard, which converts a fixed-length string into a variable-length string. A major difference between the JPEG standard and the Galileo compression scheme is that the JPEG standard uses an 8×8 discrete cosine transform (DCT). The ICT is an integer approximation of the DCT. The ICT requires only integer arithmetic, making

[1] The error correction coding, frame merging, and frame repair schemes apply to all telemetry data. The integer cosine transform (ICT) compression and error containment schemes apply to ICT image data only.

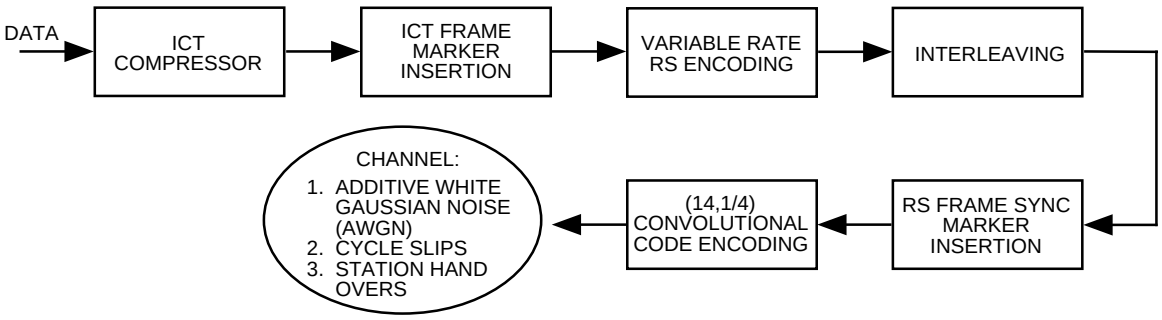

Fig. 1. Galileo spacecraft data processing system.

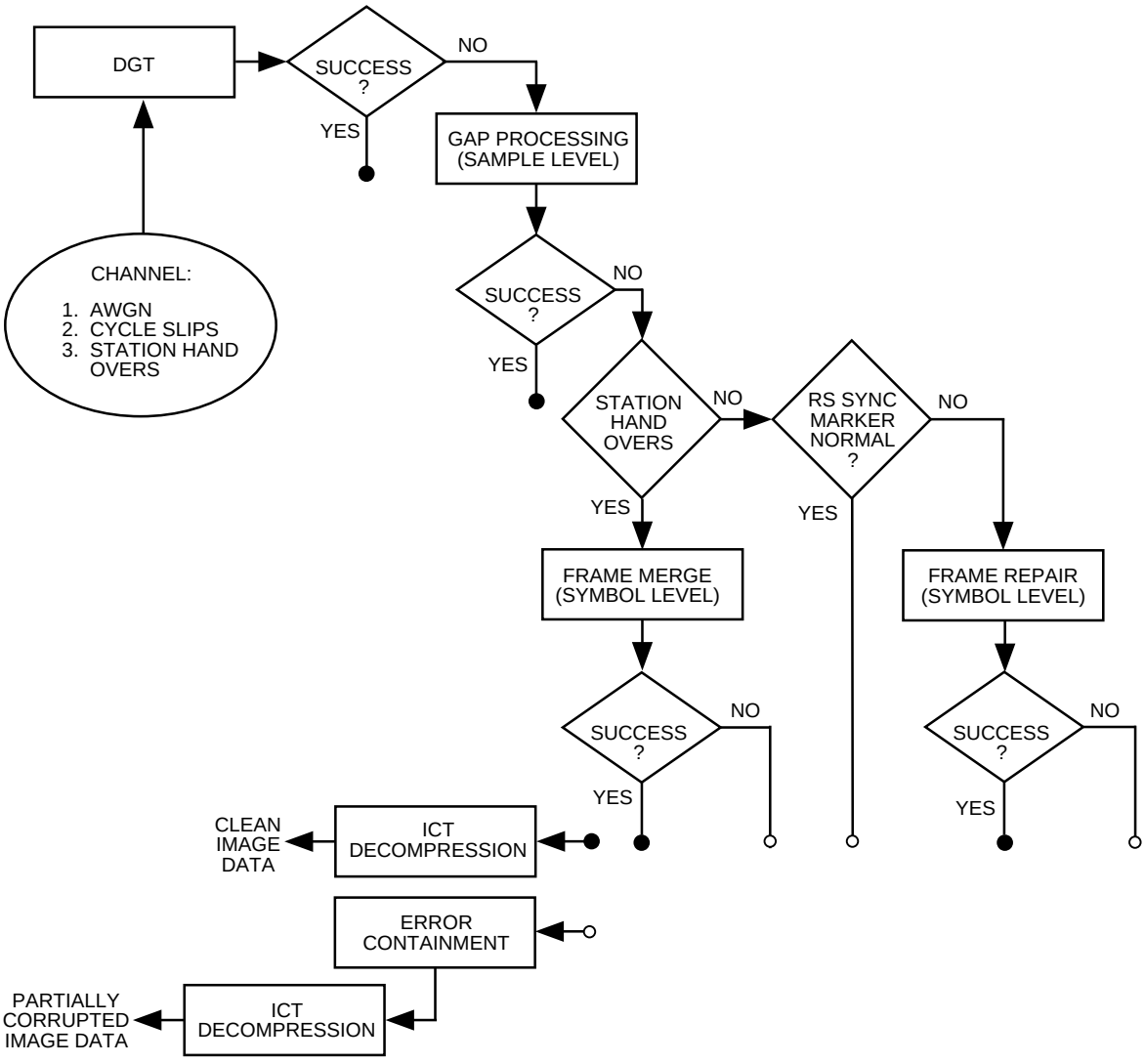

Fig. 2. Galileo ground data processing system.

it much simpler to implement. Although the ICT has a much lower complexity, rate-distortion performance is not sacrificed. This lower complexity of the ICT is necessary because the Galileo spacecraft uses late 1970s microprocessor technology. A detailed description of the Galileo ICT implementation scheme can be found in [1].

III. Error Correction

The Galileo error-correction coding scheme uses a $(255, k)$ variable redundancy Reed–Solomon (RS) code as the outer code and a (14,1/4) convolutional code as the inner code. The RS codewords are interleaved to depth 8 in a frame. The redundancy profile of the RS codes is (94,10,30,10,60,10,30,10). The staggered redundancy profile was designed to facilitate the novel feedback concatenated decoding strategy [3,4]. This strategy allows multiple passes of channel symbols through the decoder. During each pass, the decoder uses the decoding information from the RS outer code to facilitate the Viterbi decoding of the inner code in a progressively refined manner. The FCD is implemented in software on a multiprocessor workstation. The code is expected to operate at a bit signal-to-noise ratio of 0.65 dB and at a bit-error rate of 10^{-7}. Figure 3 shows the schematic of the FCD architecture. In this article, we discuss the implementation and operation aspects of the FCD task only. The FCD novel node/frame synchronization scheme is discussed in [5]. The FCD code selection and performance analysis are discussed in detail in [6].

If, due to errors beyond its capability to correct, the FCD cannot decode the data, the FCD will invoke the "frame repair" utility and the "gap processing" utility. Gap processing involves using good decoded data from either side of the gap and attempts to fill in data that are missing (the gap) because of loss of synchronization or hardware glitches. The frame repair utility attempts to correct any symbol insertions, deletions, or inversions due to transmission reception problems. The reprocessed data are then fed back to the FCD for another attempt at decoding.

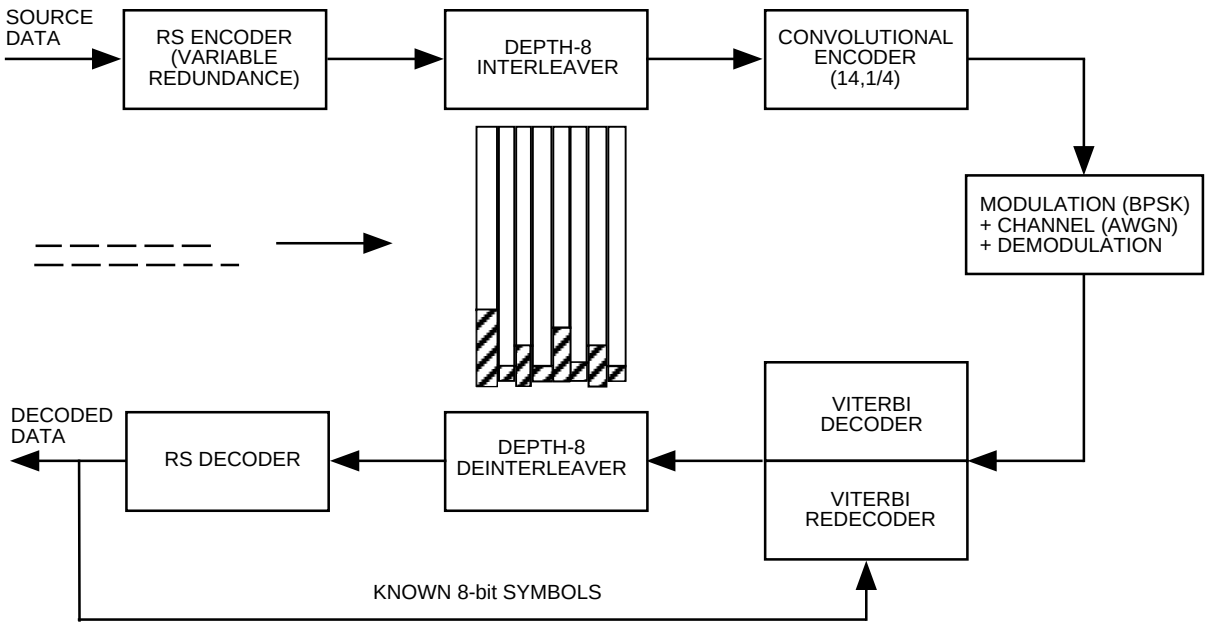

Fig. 3. FCD schematics.

A. The (255,*k*) Variable Redundancy Reed–Solomon Code

All RS codes for the Galileo mission use the same representation of the finite field $GF(256)$. To be precise, $GF(256)$ is the set of elements

$$GF(256) = 0, \alpha^0, \alpha^1, \alpha^2, \cdots, \alpha^{254}$$

where α, by definition, is a root of the primitive polynomial

$$p(x) = x^8 + x^7 + x^2 + x + 1$$

(i.e., $p(\alpha) = 0$). In the encoding/decoding process, each power of α is represented as a distinct nonzero 8-bit pattern. The zero byte is the zero element in $GF(256)$. The basis for $GF(256)$ is descending powers of α. Note that this is the conventional representation, not Berlekamp's dual basis [7]. The RS generator polynomial is defined as

$$g(x) = \prod_{i=0}^{n-k-1} \left(x - \alpha^{\beta(i+L)}\right) = \sum_{i=0}^{n-k} g_i x^i$$

where n denotes the codeword length in bytes, k denotes the number of information bytes, and α^β is a primitive element of $GF(256)$. The parameter β is chosen in some applications to minimize the bit-serial encoding complexity. Since the Galileo RS encoders are implemented in software, there is little advantage in preferring a particular value of β. The parameter L is chosen such that the coefficients of $g(x)$ are symmetrical. This reduces the number of Galois field multiplications in encoding by nearly a factor of two.

The Galileo mission utilizes four distinct RS codes. We define RS(n, k) to be an RS code that accepts as input k data bytes and produces as a codeword n bytes, where $n > k$. An RS(n, k) code can correct t errors and s erasures if $2t + s \leq n - k$. The codes are referred to as RS$(255, 161)$, RS$(255, 195)$, RS$(255, 225)$, and RS$(255, 245)$. Specifically, the parameters β and L of the four codes are

$$\begin{array}{ll} \text{RS}(255, 161) & \beta = 1, L = 81 \\ \text{RS}(255, 195) & \beta = 1, L = 98 \\ \text{RS}(255, 225) & \beta = 1, L = 113 \\ \text{RS}(255, 245) & \beta = 1, L = 123 \end{array}$$

The four RS codes, which are interleaved to depth 8, are arranged in a transfer frame as shown in Fig. 3. The RS decoders use a time-domain Euclidean algorithm to correct both errors and erasures. The details of the decoding algorithm are discussed in [8].

B. The (14,1/4) Convolutional Code and Its Parallel Viterbi Decoder

The (14,1/4) convolutional code used for the Galileo mission is the concatenation of a software (11,1/2) code and an existing hardware (7,1/2) code. The choice of convolution code is constrained by the existing (7,1/2) code, which is hardwired in the Galileo telemetry modulation unit (TMU), and by the processing

speed of the ground FCD. The generator polynomials of the (11,1/2) code and the (7,1/2) code in octal are (3403,2423) and (133,171), respectively. The generator polynomials of the equivalent (14,1/4) code are (26042,36575,25715,16723).

The Viterbi decoder for the $(14, 1/4)$ code is implemented in software in a multiprocessor workstation with a shared-memory architecture. The use of a software decoder is possible due to the slow downlink rate of the Galileo mission. The advantages of a software-based decoder are the low development cost and the flexibility allowed to perform feedback concatenated decoding. We examined two different approaches to parallelize the Viterbi algorithm: (1) state-parallel decomposition in which each processor is equally loaded to compute the add–compare–select operations per bit and (2) round-robin frame decoding that exploits the multiple processors by running several complete, independent decoders for several frames in parallel. Our early prototypes indicate that the first approach requires a substantial amount of interprocessor synchronization and communication, and this greatly reduces the decoding speed. The second approach requires minimum synchronization and communication, since each processor is now an entity independent of the other processors. The performance scaling is nearly perfect. We chose the round-robin approach for the FCD Viterbi decoder. The details of the FCD software Viterbi decoder implementation are described in [9].

IV. Frame Repair

A. Anomalous Frames

Sometimes a frame is received that has an adequate SNR, but the frame markers at the beginning and end of it have too many or too few symbols in between them and/or have opposite polarity. This indicates that some symbol insertions, deletions, and/or phase inversions, known collectively as *anomalies*, occurred somewhere in the frame. (A phase inversion at symbol x is an event where the symbols after x are detected with the opposite phase of the symbols before x. So, if the transmitted symbols were all $+1$, the received symbols, in the absence of noise, would be $+1$ up to the inversion and -1 for all symbols after that, or at least until the next phase inversion.) Because a high-redundancy Reed–Solomon codeword with too many errors is vastly more likely to be not decodable than to decode incorrectly, one can make guesses about how to fix the codeword and safely assume that a guess is correct if it results in a decodable codeword. (For Galileo's highest-redundancy Reed–Solomon codeword, which can correct up to 47 errors, if more than 47 errors occur, the probability of decoding incorrectly is $\leq 1/47!$.)

However, while guessing may be safe, it is not likely to be successful unless there is some reason for making a particular guess.

The two frame markers at either end of an anomalous frame only indicate the *most likely* situation in terms of whether or not a phase inversion occurred and how many symbols were inserted or deleted. There is always the possibility, of course, that the two markers have the same polarity because *two* inversions occurred in the frame, or that they have three more symbols than expected between them because five symbols were inserted in one area and two were deleted in another part of the frame. However, identifying and fixing (presumably rare) problems like this would be prohibitively complicated and time consuming, so the simplest explanation, namely that all anomalies occurred at the same point in the frame, is always assumed. This means that any symbols between the first and last anomaly in a frame will be useless, but if the affected interval is sufficiently small (which depends on how bad the rest of the symbols are), the errors can be tolerated.

B. Repairing Anomalous Frames

A frame that is undecodable due to anomalies in a single location can be decoded if this location is guessed correctly. A guess is tested by creating a modified version of the received frame with the appropriate number of symbols inserted or deleted, and/or with a phase inversion made, at the guessed

location. The frame is then Viterbi decoded, the result is deinterleaved, and Reed–Solomon decoding is attempted on the resulting eight words. If the highest redundancy Reed–Solomon codeword can then be decoded, it is extremely likely that the guess was approximately correct. The reason the guess need not be exact is that the effect of an incorrect guess is that essentially all of the Reed–Solomon symbols in between the actual anomaly location and the guessed one are incorrect but, if the gap is small enough, the Reed–Solomon decoder can correct those additional errors. Once the location is approximately known, the guess can be refined by trying nearby locations and choosing the one that minimizes the number of errors the Reed–Solomon decoder corrects. Then the frame, modified to reflect the refined guess, is sent on to the FCD for redecoding.

C. Finding Anomalies in a Frame

In this section, we describe the algorithm we use to find anomalies in a frame known to have some anomalies, under the assumption that they all occur at a single location. One rather simple-minded method is to try each of the 65,536 locations in the frame, one at a time. This, however, would take an unreasonable amount of time, because Viterbi decoding is slow. Slightly less simple minded is to do a binary search, since you only need to get close enough to the correct location. Thus, you first guess the location halfway through the frame, then 1/4 of the way, then 3/4, and so on. This method, in fact, is used when everything else fails, but it can still take a very long time unless you are lucky. Before resorting to this, some other educated guesses are made. If any of the following methods succeeds, meaning that the highest-redundancy Reed–Solomon codeword is decoded, the guess is then refined and the modified frame is sent to the FCD, as explained above. If not, the next method is tried. The first method: If a data rate change occurred at some point during the frame, that location is guessed, since anomalies are known to be common when data rate changes occur.[2] The second method: The adjustments are made at the end of the frame, which means that the frame is Viterbi decoded essentially as is. If the anomaly happens to be near enough to the end of the frame, this will work, but the real reason for trying this first is that the branch metrics along the decoded path are saved for use in the next step if this method does not succeed. The third method: If insertions or deletions are known to have occurred, the branch metrics mentioned above are analyzed. Although the effect is subtle when the SNR is low, the mean value of the metrics will be slightly higher after the location of the anomaly is reached than before it. This analysis results in an estimate of where this increase occurred, and this estimate is used as the next guess. The fourth method: If only a phase inversion is known to have occurred (i.e., the frame is the correct length, but the markers on either end have opposite phase), the transparency of the Galileo (14,1/4) convolutional code means that the output of the Viterbi decoder, after a transient in the vicinity of the inversion, is the same as it would have been without an inversion, except with the opposite sign. Thus, in this special case, one can make multiple guesses without having to Viterbi decode a whole frame each time. It is only necessary to invert all the Viterbi decoded symbols (from the original frame) after the guessed location and then attempt Reed–Solomon decoding. A binary search is used to make guesses throughout the frame, each of which can quickly be tested. The fifth method: The last resort is the binary search described previously. If this fails, frame repair is not accomplished.

V. Frame Merging

Because of the large frame size (65,536 symbols) and low bit rates that Galileo will be using, a single frame will take close to one-half hour to transmit at the lowest bit rate. The lowest bit rate is used near the beginning and end of a pass, when the SNR is lowest, and it is not unlikely that a frame that starts out with an adequate SNR may not be completed before the signal is lost or that it becomes noisy enough to render the whole frame undecodable. Because the ground stations have overlapping coverage, the next station may pick up the signal in the middle of the same frame. Thus, each of two stations may have an

[2] It is also observed in operation that the location in a frame when a data rate change occurs can differ from the predicts by a multiple of a 128-second span of symbols. Some frames were recovered by guessing on the 128-second boundary in the vicinity of the predicted data-rate change location.

incomplete and/or too noisy version of the same frame. Because of the nature of the encoding algorithm, a partial frame by itself is essentially useless. Since the best part of each frame is the worst part of the other, it seems obvious to try to paste them together somehow. If the SNR at the crossover point, where the falling SNR at the first station equals the rising SNR at the second, is high enough, one could make a good frame by just using the symbols from the first station up to that point and from the second station after that point. However, even if the SNR at the crossover point is significantly below the decoding threshold, by adding the symbols from the two frames together, theoretically 3 dB could be gained in the vicinity of the crossover point, assuming the noise at the two stations is independent, a reasonable assumption. Farther from the crossover point, one could still combine symbols from the two frames, but with weights reflecting their relative SNRs. Note that this requires estimates of the SNR throughout both frames and that the results can be expected to depend on the accuracy of these estimates.

The problem is made much more complicated by the fact that a low SNR not only makes symbols noisy, it can also cause anomalies, as can rate changes. In passes where both frame markers surrounding a frame are identified, the presence of anomalies is usually apparent. Notice, however, that this does not give any information about where in the frame the anomalies are. This case may be common, since frame markers can be detected at SNRs way below the threshold of decodability. In other cases, a frame may be incomplete in one or both passes or may have enough symbols for a frame but be so badly corrupted near one end that the marker cannot even be found.

A. Procedure

Here we describe the procedure that will be used to merge two undecodable versions of a frame into one decodable one for Galileo. The result of this algorithm is a small number of candidate merged frames that must then be sent to the FCD for attempted decoding.

The first step is to identify any frames that can be merged. This is done by finding frame markers near the end of the first pass and the beginning of the second one, and by the time tags, in order to determine which frames have some symbols in both passes. For each such frame, and each pass, the frame may be either partially or fully present. When an entire frame is present (in either pass), the two end markers indicate the most likely situation in terms of whether or not a phase inversion occurred and how many symbols were inserted or deleted. For the same reasons as those presented in Section IV, the simplest explanation, namely that all anomalies occurred at the same point in the frame, is always assumed.

When only part of a frame is present (in either pass), there is less information about what type of anomalies occurred. However, an estimate of the number of insertions or deletions can be made using time tags, which for Galileo occur once per second, and multiple trials can be performed for a few values near the estimate. For each value, one trial is done assuming an inversion occurred and another assuming otherwise. Notice that this procedure for finding anomalies is different from the frame repair case as it primarily depends on finding a correlation between two versions of a frame. But as in the frame repair case, the candidate modifications must be tested by attempting to decode the whole frame.

It is assumed, again because of prohibitive complexity and running time, that only one of the two passes has anomalies in it. More precisely, if a frame could have anomalies in either pass, it is first assumed that anomalies only occurred in the first pass, and next that anomalies only occurred in the second pass, with each case resulting in at least one candidate merged frame. Then, whichever frame one is assuming to be the one containing anomalies is analyzed to find the most likely location of the anomalies. This is done using dynamic programming to find the location that, if you compensate for the anomalies as if they were there, maximizes the correlation between the two frames in the region of overlap.

B. Example

Figure 4 illustrates a simplified example with 32-symbol frames (not including the marker). We will refer to the two received versions of the illustrated frame as frame A and frame B. Frame A is partial, whereas frame B is whole but clearly has two too many symbols and an inversion somewhere. Suppose that dynamic programming reveals that the best place (maximum correlation between frames A and B) in frame B to delete two symbols and invert all subsequent symbols is after symbol y_{11}. Then the candidate frame shown would be produced, where

$$z_i = \begin{cases} a_i x_i + (1 - a_i) y_i, & \text{if } i \leq 11 \\ a_i x_i + (1 - a_i)(-y_{i+2}), & \text{otherwise} \end{cases}$$

and

$$a_i = \begin{cases} \dfrac{SNR(x_i)}{SNR(x_i) + SNR(y_i)}, & \text{if } i \leq 11 \\ \dfrac{SNR(x_i)}{SNR(x_i) + SNR(y_{i+2})}, & \text{otherwise} \end{cases}$$

In practice, the SNR estimates are unlikely to be very accurate and, in any case, are not made on a symbol-by-symbol basis, but rather over, perhaps, a minute; the algorithm uses whatever it gets.

In this example, additional candidate frames would be produced, based on the assumption that frame B is anomaly free and frame A may not be. Even though we know frame B has anomalies, this may be helpful if frame A also has anomalies and frame B's anomalies all happen to be very close to the beginning marker, which is plausible since the SNR is probably lowest there. The "anomaly free" frame B is obtained by inverting all the symbols in the frame and throwing out y_1 and y_2. An estimate of the number of symbol insertions or deletions in frame A is obtained by using the time tags to estimate the time t of symbol x_{21} and by finding i such that t falls between the estimated times of symbols y_i and y_{i+1} (t_1 and t_2, respectively.) For instance, suppose that $i = 24$, with t closest to t_1. Then the most likely case is that frame A is missing one symbol, and the next most likely is that it is missing two, and both hypotheses will be used to generate candidate frames.

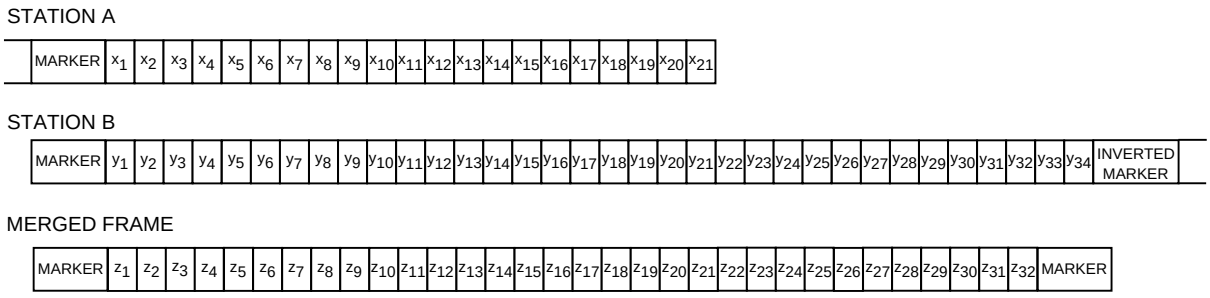

Fig. 4. Frame merge example.

VI. Error Containment

If all these attempts fail to correct the data, then the final step is to minimize error propagation. Because any corruption of data in a compressed data stream can cause magnified errors due to propagation, safeguards were introduced into the compressed data stream to minimize these effects. First, images are compressed independently by groups of 8 rows, which we called slices, with a synchronization marker inserted at the beginning, so any bit errors during data transmission can be isolated. Second, the decompression algorithm includes provisions for detection of errors so that the error containment mode will be activated.

These two safeguards are used together in the following manner: If an error is detected during decompression, the decompression algorithm will start a search for the synchronization marker. This marker is 32-bits wide and consists of 2 fields: a 7-bit sequencing field and a 25-bit fixed-pattern field. The 7-bit field will identify where the slices belong relative to the whole image, and the 25-bit field is used for synchronization. Using these safeguards, error propagation can at most be isolated to 8 rows of image data.

Figures 5 and 6 show two ICT compressed/reconstructed images of Ganymede (moon of Jupiter) that are corrupted with random errors in the compressed data of a bit-error rate (BER) of 0.0001 and 0.001, respectively. In both images, the ICT error containment scheme detects the onset of the uncorrectable errors, prevents them from propagating beyond 8 rows, and automatically resynchronizes the good data thereafter. Thus, the majority of the image areas remain intact and can still be used for scientific investigation.

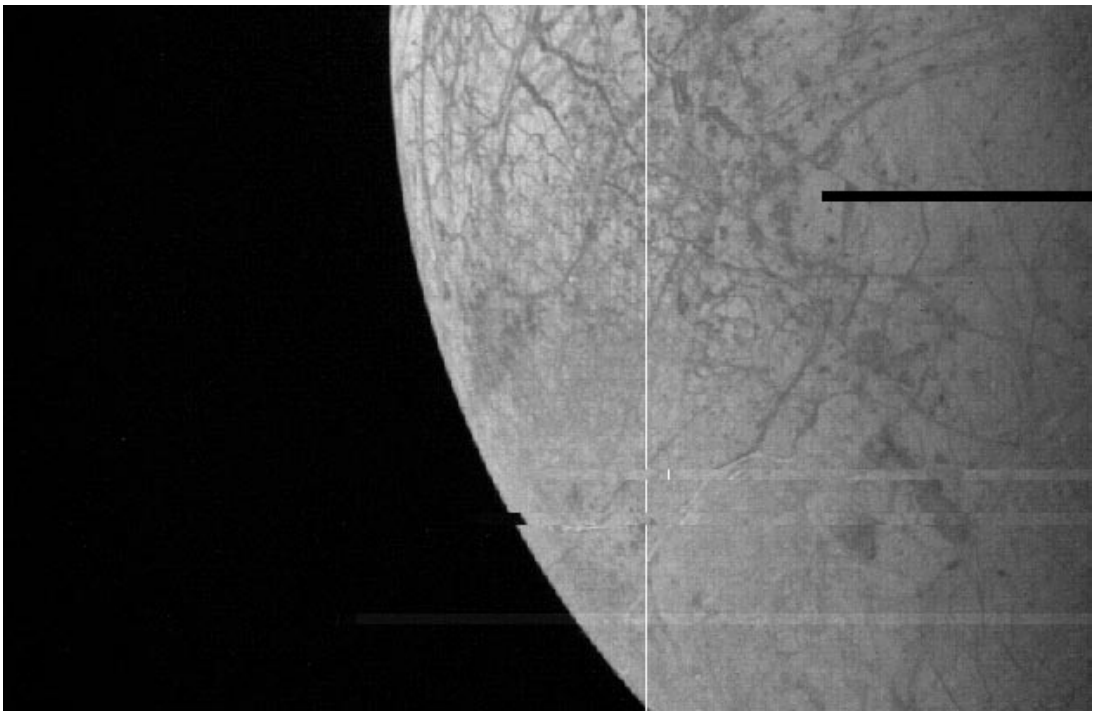

Fig. 5. Ganymede with BER = 0.0001.

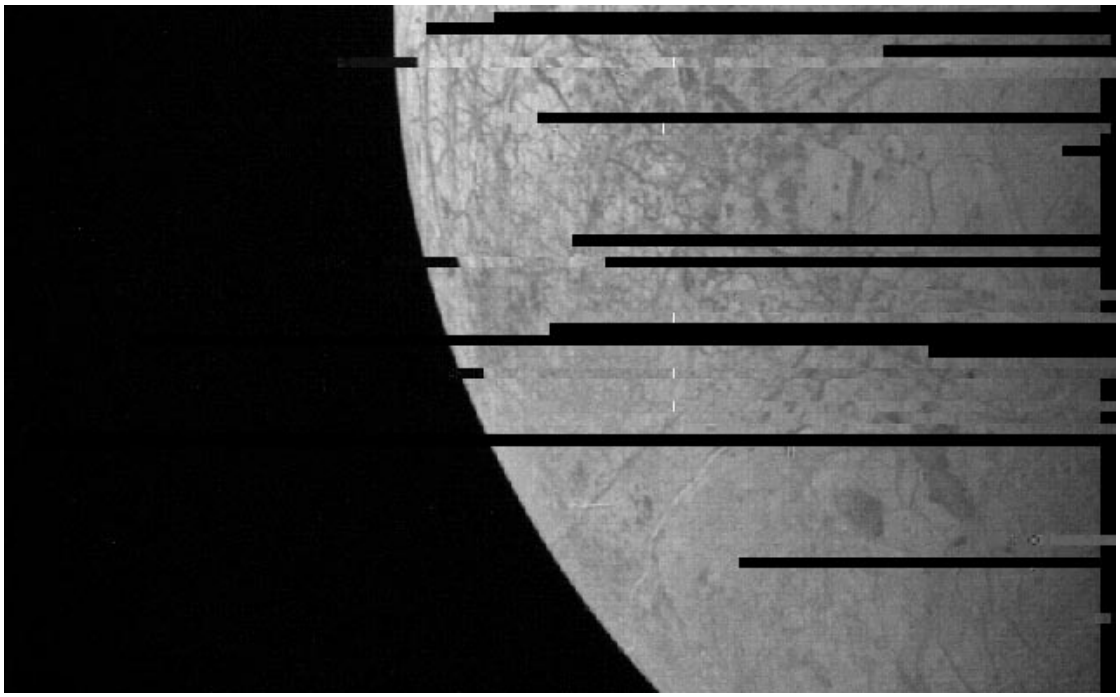

Fig. 6. Ganymede with BER = 0.001.

Acknowledgment

A part of this article was presented and published at the 1996 International Geoscience and Remote Sensing Symposium (IGARSS'96) in Lincoln, Nebraska, May 23–31, 1996.

References

[1] K. Cheung and K. Tong, "Proposed Data Compression Schemes for the Galileo S-Band Contingency Mission," *Proceedings of the 1993 Space and Earth Science Data Compression Workshop*, Snowbird, Utah, pp. 99–110, April 2, 1993.

[2] W. Cham, "Development of Integer Cosine Transform by the Principle of Dyadic Symmetry," *IEE Proceedings*, vol. 136, pt. I, no. 4, August 1989.

[3] E. Paaske, "Improved Decoding for a Concatenated Coding System Recommended by CCSDS," *IEEE Transaction of Communications*, vol. COM-38, August 1990.

[4] O. Collins and M. Hizlan, "Determinate-State Convolutional Codes," *The Telecommunications and Data Acquisition Progress Report 42-107, July–September 1991*, Jet Propulsion Laboratory, Pasadena, California, pp. 36–56, November 15, 1991.

[5] J. I. Statman, K.-M. Cheung, T. H. Chauvin, J. Rabkin, and M. L. Belongie, "Decoder Synchronization for Deep Space Missions," *The Telecommunications and Data Acquisition Progress Report 42-116, October–December 1993*, Jet Propulsion Laboratory, Pasadena, California, pp. 121–127, February 15, 1994.

[6] S. Dolinar and M. Belongie, "Enhanced Decoding for the Galileo Low-Gain Antenna Mission," *Proceedings of the 1994 IEEE International Symposium on Information Theory*, Trondheim, Norway, June 27–July 1, 1994.

[7] E. Berlekamp, "Bit-Serial Reed–Solomon Encoder," *IEEE Transaction of Information Theory*, vol. 28, 1982.

[8] R. McEliece, "The Decoding of Reed–Solomon Codes," *The Telecommunications and Data Acquisition Progress Report 42-95, July–September 1988*, Jet Propulsion Laboratory, Pasadena, California, pp. 153–167, November 15, 1988.

[9] T. Chauvin and K. Cheung, "A Parallel Viterbi Decoder for Shared Memory Architecture", presented at the SIAM Conference on Parallel Signal/Image Processing on Multiprocessor Systems, Seattle, Washington, August 1993.