BIT ERROR DETECTION AND RECOVERY FOR X2D MMR CODED BITSTREAMS

Hyunju Kim and Abdou Youssef
Department of Computer Science, The George Washington University
{hkim, youssef}@seas.gwu.edu

ABSTRACT

This paper proposes a bit error recovery method for Extended 2 Dimensional MMR coded bitstreams. When an error occurs in an MMR coded bitstream, the bitstream cannot be decoded correctly after the error point. To prevent losing valid information after an error, we developed an error recovery system that detects bit errors and applies bit-inversion to correct the errors. In case the bit-inversion cannot correct the error, the system applies new algorithms that utilize syntactical structure information of the coded bitstream to recover nearly all the data.

1. INTRODUCTION

The amount of data traffic over communication lines is vast and growing daily, in part because images consist of massive volumes of data. As a result, images are often compressed. Since nearly all communication media are noisy and incur error, the impact of noise is very serious on compressed data and the errors are hard to detect and recover from. Yet without error recovery, compressed data cannot be decoded.

Most receivers solve this problem by requesting retransmission. But this adds more network traffic, requires more time, and incurs additional costs. Even worse, retransmission is not possible in surveillance applications. In case retransmission is not desirable or possible, the decoder should try to recover as much lost data as possible with received data only.

Most of the research on error recovery has focused on error concealment methods for lossy block-based DCT compression as in JPEG or MPEG [1,2,7]. These methods assume that a bitstream has been decoded and the position of the erroneous block is known. For error recovery in losslessly compressed bitstream, which is relevant to this paper, relatively little research has been done [6,8]. In [8], a generic error recovery scheme is presented where the error patterns are assumed to be known, which is the case in the older modems. In [6], an error recovery method is presented for facsimile image without assuming any knowledge of error patterns, but the image in their research is not compressed in MMR, which is used in most current facsimile machines. Our research is the first to address error recovery in MMR coded bitstreams. In two other papers [4,5], we developed symbol error recovery in MMR fax over modems. In this paper, we address single-bit error recovery in MMR. We define a single-bit error as an isolated wrongly flipped bit, which usually occurs in fax over IP. Thus, this paper assumes the following error model:

- There is no retransmission.
- There is no error resiliency tool or code provided by the encoder or network channel.
- Bitstreams are coded with MMR code.
- The errors are single-bit errors.

The next section provides a brief overview of MMR coding. Error detection, resynchronization conditions, and our error recovery approach are presented in section 3. Section 4 illustrates the performance of the approach, and section 5 provides conclusions.

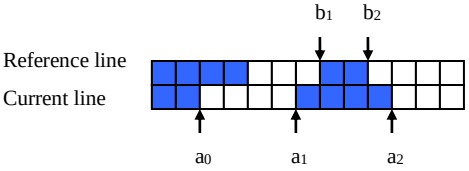
2. X2D MMR CODE

The majority of facsimile machines in the world apply Extended 2 Dimensional (X2D) MMR (Modified Modified READ) coding to compress facsimile images. This scheme is used for Group 3 and Group 4 facsimile images and defined in ITU (formally CCITT) T.4 and T.6 recommendations [3].

MMR makes use of vertical relationships between black runs and white runs in two adjacent scan lines. It codes an image one scan line (1728 pixels) at a time and uses the previous line to code the current scan line. Figure 1 shows the elements of MMR coding.

In MMR, when coding the topmost line, the Reference line is assumed to be an imaginary blank line above the page. Also, when coding a Current line, the initial value of the first changing element is taken to be an imaginary blank pixel left of the line. The position of a_0 changes according to the rules in Figure 1, and the position of a_1, a_2, b_1, and b_2 change in a way that is consistent with their definitions relative to a_0.

When an error occurs in an MMR coded bitstream, the error propagates through the bitstream because MMR coding does not provide any resynchronization point. As a result, the decoder produces a corrupted image beyond the error position.



a_0: The position of the first changing element on the current line.
a_1: The next changing element to the right of a_0 on the current line.
a_2: The next changing element to the right of a_1 on the current line.
b_1: The first changing element in the reference line to the right of a_0 and of opposite color of a_0.
b_2: The next changing element to the right of b_1 on the reference line.

(a) Changing picture elements

Coding Modes	Conditions	Codes	Next values of a_0
Pass Mode (P Mode)	$a_1 > b_2$	0001	$a_0 := b_2$
Horizontal Mode (H Mode)	$a_1 \le b_2$ & $\lvert a_1 - b_1 \rvert > 3$	001 + M*(a_0a_1) + M*(a_1a_2)	$a_0 := a_2$
Vertical Mode (V Mode)	$a_1 \le b_2$ & $\lvert a_1 - b_1 \rvert \le 3$		
V(0)	$a_1 = b_1$	1	$a_0 := a_1$
$V_L(1)$	$a_1 = b_1 - 1$	010	$a_0 := a_1$
$V_L(2)$	$a_1 = b_1 - 2$	000010	$a_0 := a_1$
$V_L(3)$	$a_1 = b_1 - 3$	0000010	$a_0 := a_1$
$V_R(1)$	$a_1 = b_1 + 1$	011	$a_0 := a_1$
$V_R(2)$	$a_1 = b_1 + 2$	000011	$a_0 := a_1$
$V_R(3)$	$a_1 = b_1 + 3$	0000011	$a_0 := a_1$

* $M(a_0a_1)$ and $M(a_1a_2)$ are Huffman codewords of the lengths $(a_1 - a_0)$ and $(a_2 - a_1)$. Two Huffman tables are specified in T.4/T.6, one for white runs and one for black runs.

(b) MMR coding and update of a_0

Figure 1 MMR coding

3. OUR APPROACH TO BIT ERROR RECOVERY

Our approach to bit error recovery is to (1) detect the error, (2) determine the region of error, and (3) apply bit inversion and re-decoding to bits within the region. If the third step fails to recover an image from error, our system tries to find a resynchronization point in the bitstream to resume decoding.

3.1 Error detection

To detect an error, we use the constraints implied in MMR shown in Figure 1.

Each MMR code mode has conditions. When our error recovery system finds any inconsistencies between a code mode and the corresponding conditions, it declares that an error occurred in the bitstream. Specifically, our error detection is done by checking for the following violations of the constraints:

- No codeword in the code table matches the received bit pattern.
- The Pass mode occurs, but the changing element b_2 is off the right end of the scan line.
- $V_R(i)$, $i = 1$, or 2, or 3 occurs, but the changing element b_1 is off the right end of the scan line.
- $V_L(i)$, $i = 1$, or 2, or 3 occurs, but the changing element are such that $b_1 - a_0 - i \le 0$.
- The Horizontal mode occurs and the run-length a_0a_1 is decoded, but the changing elements are either $| a_1 - b_1 | \le 3$ or $b_2 < a_1$.
- The Horizontal mode occurs and the run-lengths a_0a_1 and a_1a_2 are decoded, but the a_1a_2 is off the right end of the scan line.
- The Horizontal mode occurs and the run-length a_0a_1 is decoded, but the corresponding a_1a_2 does not exist.
- The Horizontal mode occurs and the color of the run-length a_0a_1 is determined, but there is no codeword in the color runs table that matches the received bit pattern.

3.2 Error region determination

An error detection point, denoted by ED_i, occurs after the error position, denoted by E_i. The distance $ED_i - E_i$ is called the *detection latency*. The region of error ranges from E_i to ED_i ($E_i < ED_i$) in the bitstream. However, the exact position of the error, E_i, is not known when the error is detected. Thus, we need to find an earlier start point, denoted by ES_i, of the error region ($ES_i < E_i < ED_i$). Thus, practically, the region of error ranges from ES_i to ED_i in the bitstream. The length $ED_i - E_i$ of the error region has been estimated experimentally to be 815 bits, as discussed in Section 4. Thus, ED_i is found by the error detection module, and $ES_i = ED_i - 815$, leading to a full determination of the error region.

3.3 Error recovery

Once an error region is determined, bit-inversion and re-decoding is applied to every bit in the region. The system checks for the violations of the constraints during the re-decoding step to see whether the bit-inversion is valid or not. If there are any violations, the system re-instates the bit and moves to the next bit. After all bits in the region have been tried, if the error persists, the system assumes that there are more than one bit error or the error occurs outside of the presumed error region. In this case, it finds a resynchronization point S_i ($ES_i < E_i < ED_i < S_i$) to resume decoding.

In this approach, the problem is that MMR coding does not provide any synchronization codeword. Thus, we must find a portion of the bitstream that can be used somehow as a synchronization point. The next sub-

section explains our approach to locate a synchronization point within an MMR coded bitstream.

3.4 Resynchronization in MMR

A bitstream portion should satisfy one of the following two conditions in order to provide a synchronization point:

(1) A portion has a guessable Reference line, or

(2) A portion needs no Reference line for decoding.

Derived from the syntactical information in MMR, two types of scan lines can be used as a synchronization point: a *white line* (blank line) and a line consisting of the Horizontal modes only (*all-H line*). A white line, which satisfies the first condition, can be used as a Reference line for the next line since it consists of only white pixels. Then, decoding can be resumed from the following line. An all-H line does not refer to the previous line since the Horizontal mode explicitly codes the run-lengths of white runs and black runs, thus it satisfies the second condition.

For each type of the scan line to be used as a resynchronization point, we determined a regular expression (pattern) to search for in the bitstream to locate such a resynchronization point, as explained next.

According to MMR, a white line following a non-white line is coded with one or more Pass modes followed by one *V(0)* mode: *P … P V(0)*, represented by the regular expression $P^+ V(0)$. Each white line that follows a white line is coded as *V(0)*. The first non-white line after the white lines is coded by one or more Horizontal modes followed by one Vertical mode, which is represented by a regular expression $H^+ V_D(i)$, where *D* is either *L* or *R* and *i = 0*, *1*, *2*, or *3*. In summary, the first type of synchronization point, called *white line resynchronization*, is accomplished by searching in the bitstream for a regular expression of the form $P^+ (V(0))^+ H^+ V_D(i)$.

An all-H line is represented by *H … H* (that is, H^+). But the pattern is extremely rare since most of the binary images have right and left margins that are represented by Vertical modes in MMR code. Consequently, the expression can be modified by adding two Vertical modes at the beginning, so $V_D(i)\ V_D(j)\ H^+$ will be reflecting the margins, where *D* is *L* or *R*, and *i, j = 0*, *1*, *2*, or *3*. $V_D(i)$ represents the right margin of the previous non-all-H line, and $V_D(j)$ represents the left margin of the current all-H line. Thus, the second type of synchronization point, called *all-H line resynchronization*, is accomplished by searching in the bitstream for the regular expression $V_D(i)\ V_D(j)\ H^+$.

These resynchronizations are used when the error recovery system fails to correct bit error(s) with the bit-inversion. The system calls the resynchronization modules to find a synchronization point so that it skips the error region and keeps decoding.

The white line resynchronization works very well for text images because usually many successive blank lines intervene between non-blank lines. As long as a binary image has a blank line, our system is able to resynchronize decoding by the line. The all-H line resynchronization applies well when a binary image has graphic contents.

4. PERFORMANCE EVALUATION

The error recovery system has been implemented on a Pentium 4, 1.6 GHz using the C programming language. We evaluated the system with 7 facsimile test images provided by ITU, using what we call the Relative Error (RE) metric, defined as

$$RE = \frac{E_o / S_o}{E_b}$$

where E_o is the total number of erroneous binary pixels in the output image, S_o is the size of the uncompressed image, and E_b is the number of error bits in the bitstream. RE ranges from 0 (best) to 1 (worst).

Our test results show that the average detection latency, denoted by DL_μ, is about 80 bits, and the standard deviation of the detection latency, denoted by DL_σ, is about 245 bits. This means when error bits are more than 80 bits apart, which is the case when the Bit Error Rate (BER) is less than 1.25%, recovery is 100% successful on average. It also means that by setting the error region length to $DL_\mu + 3DL_\sigma = 815$ bits, there is an estimated 99.87% chance that the error bit will be found and corrected in that region (assuming the detection latency follows a normal probability distribution).

Indeed, we have found that about 90.5% of the test cases successfully corrected errors with the bit inversion (RE = 0). About 8% of the test cases did not correct errors, but passed the re-decoding checking without any violation detection and produced decoded images where the error is barely noticeable in the output. The RE ranges from 2×10^{-6} to 7×10^{-4}, and averages at 10^{-4}. On the other hand, assuming no error recovery for the same test cases, the RE increases to the range from 0.0014 to 0.2, and averages at 0.04. The remaining 1.5% of the test cases failed to recover from errors with bit inversion.

When bit inversion cannot recover an image, the resynchronization module is applied to the image. Figure 2 shows an example of the resynchronization approach. Figure 2(b) shows the effect of no recovery: the image has nothing beyond the error detection point when no error recovery is applied. Figure 2(c) presents a recovered image with our white line resynchronization module. Obviously, the recovered image has a much smaller RE value compared to the value of the remaining image.

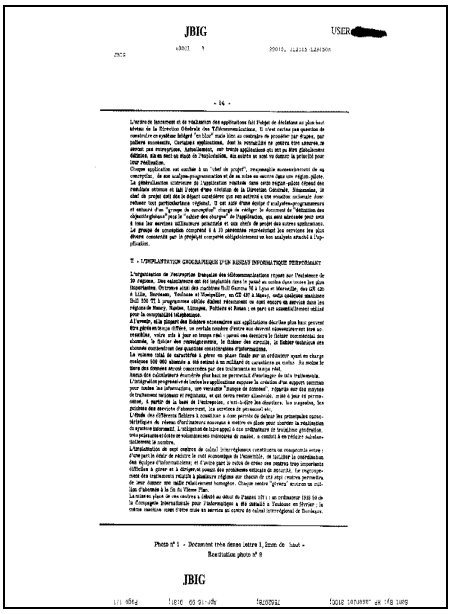
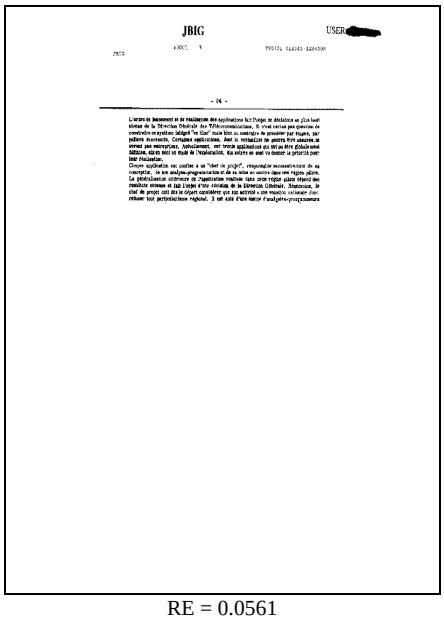


Figure 2(a) Original ITU test image

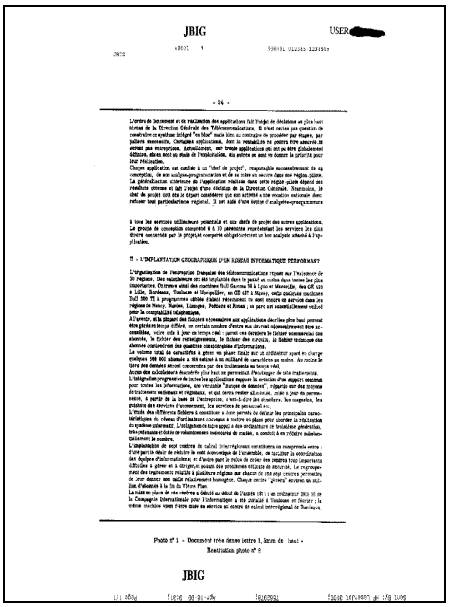


RE = 0.0561

Figure 2(b) Decoded image without error recovery

5. CONCLUSION

In this paper, a bit error recovery approach for MMR coded bitstreams is proposed. We developed an error recovery system, which detects an error and applies bit-inversion and re-decoding to correct error(s) within the error region. If the error cannot be corrected with this step, it finds a resynchronization point from which it resumes decoding.



RE = 0.0026

Figure 2(c) Recovered image with resynchronization

The test results show that our approach successfully recovers images from bit errors when BER is less than 1.25%. In case BER is greater than 1.25%, our system calls the resynchronization modules that also recover images rather well by locating specific patterns of codewords within MMR coded bitstreams.

6. REFERENCES

[1] Alkachouh, Z., and Bellanger, M., "Fast DCT-Based Spatial Domain Interpolation of Blocks in Images", *IEEE Trans. on Image Processing*, vol. 9, no. 4, Apr. 2000, pp. 729-732.

[2] Hemami, S., and Meng, T., "Transform Coded Image Reconstruction Exploring Interblock Correlation", *IEEE Trans. on Image Processing*, vol. 4, no. 7, Jul. 1995, pp. 1023-1027.

[3] ITU-T Recommendation T.6, Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus, 1988.

[4] Kim, H., and Youssef, A., "Error Recovery in Facsimile without Retransmission", accepted to appear in CATA2003, Mar. 26-28, Honolulu.

[5] Kim, H., and Youssef, A., "Interactive Error Recovery in Facsimile without Retransmission", accepted to appear in ITCC2003, Apr. 28-30, Las Vegas.

[6] Shyu, W., and Leou, J., "Detection and Correction of Transmission Errors in Facsimile Images", *IEEE Trans. on Communications*, vol. 44, no. 8, Aug. 1996, pp. 938-948.

[7] Wang, Y., Zhu, Q., and Shaw, L., "Maximally Smooth Image Recovery in Transform Coding", *IEEE Trans. on Communications*, vol. 41, no. 10, Oct. 1993, pp.1544-1551.

[8] Youssef, A., and Ratner, A., "Error Correction in HDLC without Retransmission", In Proceedings of CISST, vol. 2, 2001, pp. 526-532.