# A Hyper-Trellis based Turbo Decoder for Wyner-Ziv Video Coding

Arun Avudainayagam, John M. Shea and Dapeng Wu
Wireless Information Networking Group (WING)
Department of Electrical and Computer Engineering
University of Florida
{arun@dsp, jshea@ece, wu@ece}.ufl.edu

***Abstract*— A new approach to design video coding schemes for wireless video applications has emerged recently. Schemes using this approach are based on the principle of distributed source coding, and they exploit the correlation in the frames of the video sequence at the decoder. One such scheme models the correlation between frames as a *Laplacian* channel, and uses a turbo code to correct errors occurring in this pseudo-channel. Existing implementations use a sub-optimal approximation to compute the channel likelihoods in the BCJR algorithm in the turbo decoder. To resolve this sub-optimality, we propose applying the BCJR algorithm on a trellis-structure with parallel transitions. This trellis structure is called a *hyper-trellis*. The BCJR maximum *a posteriori* algorithm is modified to use the hyper-trellis, and computation of the channel-likelihoods are shown to be optimal on the hyper-trellis. Simulation results show that the hyper-trellis approach can yield a $5$ dB improvement in peak signal-to-noise ratio.**

## I. Introduction

In traditional video coding schemes like *MPEG* and *H.*$26x$, the encoder bears most of the computational burden when compared to the decoder. The statistical correlation in the frames of the video sequence is exploited at the encoder to perform predictive coding. Predictive coding is usually performed by using motion estimation algorithms that are computationally expensive. However, the advent of wireless video and sensor networks have placed stringent requirements on the complexity of the video encoder. In these applications, video coding has to be performed in small, power-constrained, and computationally-limited low-cost devices. These applications call for a simple encoder to reduce cost of the video sensing device, and to enable real-time encoding. Though a simple encoder is required, the coding rate should not be compromised because this directly impacts the amount of power consumed in transmission.

Low-complexity codecs based on the principles of distributed source coding have been proposed recently for wireless video applications [1], [2] . These techniques are similar in the fact that they exploit source statistics at the decoder. The Slepian-Wolf theorem [3] and its continuous-source counterpart, the Wyner-Ziv result [4] motivate this approach. These information-theoretic results state that it is possible to encode the frames of a video sequence (or any correlated source) independently and still achieve efficient compression as long as decoding is performed jointly. In the context of video coding these results imply that (ideally) all the processing to exploit temporal (interframe) and spatial (intraframe) correlation in the video stream should be performed the decoder. This facilitates the design of a simple video encoder at the cost of increased complexity at the decoder.

In this paper, we focus on the scheme introduced by Aaron and Girod [1], [5]. In this scheme, the frames are encoded independently (intraframe encoding) and decoded jointly (interframe decoding). Since their technique is based on the Wyner-Ziv theorem [4] on source coding with side-information at the decoder , we refer to their codec as the Wyner-Ziv video codec. The decoder is assumed to have side-information (SI) about the frames to be encoded. The correlation between the pixels in the SI frame and the original frame is modelled as a Laplacian channel. A turbo code is then used to correct the errors in this “correlation-channel”. It will be shown in the next section that the channel likelihoods in the turbo decoder are computed in a sub-optimal manner in  [1], [5]. This limits the application of the Wyner-Ziv codec to situations in which the quality of the SI is good (low mean-squared-error between the SI and the original frame). In this paper, we introduce a new trellis structure (a *hyper-trellis*) for use in the Wyner-Ziv video decoder. By using this hyper-trellis, the channel-likelihoods can be optimally calculated using a modified version of the BCJR [6] algorithm. The decoder complexity does not increase if the hyper-trellis is used for decoding. It will be shown in Section IV that the hyper-trellis approach can increase the peak-signal-to-noise ration by more than $5$ dB when compared to the approach in [1], [5].

## II. Wyner-Ziv video coding using turbo codes

In this section, we briefly explain the operation of the Wyner-Ziv video codec presented in [1], [5]. We also discuss in some detail the BCJR maximum *a posteriori* (MAP) decoding [6] algorithm used in the Wyner-Ziv decoder. Although this is a well-established scheme, repeating some of the equations here makes it easier to identify the sub-optimality of the technique in [1], [5]. These equations also facilitate the development of the BCJR algorithm on the hyper-trellis (Section III-B).

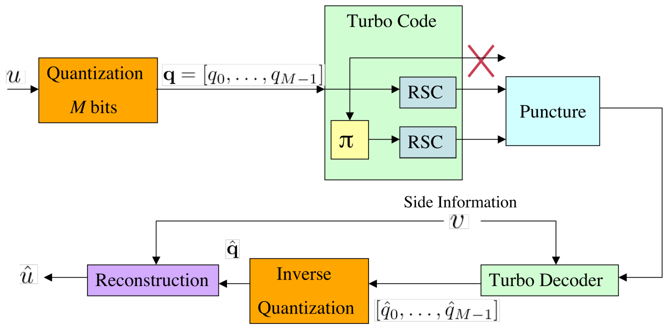

Fig. 1. Wyner-Ziv Codec

The Wyner-Ziv codec of Aaron and Girod operates as follows. Let $F_1, F_2, \ldots, F_N$ be the frames of the video sequence. The odd frames ($F_{2i+1}$) of the video sequence are intra-coded using the typical approach of applying a discrete cosine transform (DCT) followed by uniform quantization with different step sizes for each DCT coefficient. These intra-coded frames are referred to as $I$-frames in *MPEG* and *H.26x* video coding schemes. The $I$-frames can be decoded with high accuracy. At the decoder, the odd frames are used to generate side-information (SI) ($S_{2i}$) for the even frames ($F_{2i}$). Thus, the $I$-frames are also referred to as key frames. The SI $S_{2i}$ is usually generated by interpolating the key frames. Following the approach of [1], [5], we do not consider coding of the odd frames, and assume that perfect estimates of the key frames are available to the decoder.

The Wyner-Ziv codec is shown in Figure 1. The Wyner-Ziv codec is only used to encode and decode the even frames. Each pixel $u$ of $F_{2i}$ is quantized using a uniform quantizer with $2^M$ levels to produce a quantized symbol[1] $\mathbf{q}$. The quantized symbol $\mathbf{q}$ is then converted to a binary codeword $[q^0, \ldots, q^{M-1}]$. A sufficient number of symbols are collected and are encoded using a turbo code. The turbo encoding is done as follows. The quantized symbols are converted to binary codewords and encoded using a recursive systematic convolutional (RSC) code to produce a systematic bit stream and a parity bit stream. The quantized symbols are interleaved on the symbol level and then encoded using another identical RSC code to produce a second parity bit stream.

The decoder has SI ($S_{2i}$) about the current frame $F_{2i}$ . In the video coding literature, the difference between the pixels of $F_{2i}$ and $S_{2i}$ is usually modeled as a Laplacian random variable. I.e.,

$$\{S_{2i}\}_{m,n} = \{F_{2i}\}_{m,n} + \eta, \tag{1}$$

where $(m, n)$ indexes the pixel location, and $\eta$ is a Laplacian random variable with density function $p(\eta) = \frac{\alpha}{2} e^{-\alpha|\eta|}$. Thus, $S_{2i}$ can be considered to be the output of a channel with Laplacian noise to which the original frame $F_{2i}$ is the input. This fictitious channel that arises because of the correlation between the frames of the video is sometimes referred to as the *correlation* channel. Since a noisy version of $F_{2i}$ is available at the decoder in the form of $S_{2i}$, the systematic part of the turbo code need not be transmitted. As shown in Figure 1, the systematic part is discarded. The parity streams generated by the two RSCs can be punctured to obtain any desired rate. Compression is achieved when fewer parity bits are transmitted than the size of the input bit stream to the turbo encoder. The turbo decoder in the receiver has to estimate the original frame $F_{2i}$ from the SI $S_{2i}$ and the parity bits. The parity bits are assumed to be transmitted through an error-free channel. This is a common assumption in source coding, wherein the compressed bits are assumed to be error-free at the source decoder. In [1], [5], a feedback channel is assumed between the decoder and the transmitter. The transmitter starts by sending a small set of parity bits. If the decoder encounters a bit error rate greater than a pre-detemined threshold after decoding, it requests additional parity bits from the transmitter. Parity bits are requested until a target bit error rate is acheived.

The Wyner-Ziv video decoder consists of a turbo decoder followed by a reconstruction function. The Wyner-Ziv decoder estimates the pixels of the original frame ($F_{2i}$) in a two-step process. First, the turbo decoder operates on the transmitted parity bits along with the side-information to produce an estimate of the quantized symbols, $\hat{\mathbf{q}}$. This estimate $\hat{\mathbf{q}}$, and the SI $S_{2i}$ are used to reconstruct an estimate of the original pixel $u$ in frame $F_{2i}$ as $\hat{u} = E(u|\hat{\mathbf{q}}, S_{2i})$. Details about implementing the reconstruction function can be found in [1].

## A. BCJR MAP decoding on the regular trellis

We now modify the BCJR MAP algorithm that is used in the the turbo decoder to the context of Wyner-Ziv video coding. This is a simple extension of the BCJR MAP decoder for additive white Gaussian noise channels presented in [7]. The following notation is used. The input to the rate-1/2 RSC code is represented as $\mathbf{x} = [x_1, \ldots, x_K]$. The output of the RSC code is denoted by $\mathbf{c} = [\underline{c}_1, , \ldots, \underline{c}_K]$, where each $\underline{c}_i = [x_i, p_i]$ where $p_i$ is the parity bit produced by the RSC encoder at time $i$. The received vector at the decoder is represented by $\mathbf{y} = [\underline{y}_1, \ldots, \underline{y}_K]$, where each $\underline{y}_i = [y_i^x, y_i^p]$. We have used $y_i^x$ to represent the received value for the systematic bit $x_i$ and $y_i^p$ to represent the received value for the parity bit $p_i$. The state of the encoder at time $i$ is denoted by $s_i$.

The BCJR MAP decoder computes the log-likelihood ratio (LLR) for information bit $x_k$ as

$$L(x_k) = \log \frac{P(x_k = 0|\mathbf{y})}{P(x_k = 1|\mathbf{y})}. \tag{2}$$

The decoder decides $\hat{x}_k = 0$ if $L(x_k) > 0$, and $\hat{x}_k = 1$ if $L(x_k) < 0$. Following the development in [7], the LLR can

[1]Note that the quantized symbol ($\mathbf{q}$) is not a vector. We use the vector notation to indicate that the quantized symbol has a binary representation given by $\mathbf{q} = [q^0, \ldots, q^{M-1}]$. This notation allows us to use the quantized symbol ($\mathbf{q}$) and the binary representation of the quantized symbol ($[q^0, \ldots, q^{M-1}]$) interchangably, thereby simplifying exposition.

be expressed as

$$L(x_k) = \log\left(\frac{\sum_{\mathcal{X}_0} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}{\sum_{\mathcal{X}_1} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}\right), \quad (3)$$

where $\mathcal{X}_0$ is the set of all transitions from state $(s_{k-1} = s') \rightarrow (s_k = s)$ with an input label of 0, $\mathcal{X}_1$ is similarly defined, and the branch metrics $\alpha_k(s)$, $\beta_k(s)$ and $\gamma_k(s',s)$ are defined as follows:

$$\alpha_k(s) \triangleq P(s_k = s, \mathbf{y}_1^k) = \sum_{s_{k-1}=s'} \alpha_{k-1}(s')\gamma_k(s',s), \quad (4)$$

$$\beta_k(s) \triangleq P(\mathbf{y}_{k+1}^K | s_k = s) = \sum_{s_k=s} \beta_k(s)\gamma_k(s',s), \quad (5)$$

$$\gamma_k(s',s) \triangleq P(s_{k=s}, \underline{\mathrm{y}}_k | s_{k-1} = s'). \quad (6)$$

The branch metric $\gamma_k(s',s)$ can be further reduced to [7],

$$\gamma_k(s',s) = P(x_k)P(y_k^x|x_k)P(y_k^p|p_k). \quad (7)$$

Note that $P(x_k)$ denotes the *a priori* probability of $x_k$. This quantity takes the value of the extrinsic information at the other constituent decoder (see [7]). $P(y_k^p|p_k)$ is the likelihood for the parity bits. In the Wyner-Ziv video coding setup, the parity bits at the output of the encoder are either punctured, or transmitted to the decoder without errors. Thus, the likelihoods of the parity bits can be evaluated as

$$P(y_k^p|p_k) = \begin{cases} 1, \ y_k^p = p_k, \ p_k \text{ not punctured} \\ 0, \ y_k^p \neq p_k, \ p_k \text{ not punctured} \\ 0.5, \ p_k \text{ punctured} \end{cases} \quad (8)$$

The probability $P(y_k^x|x_k)$ in (7) is sometimes called the $channel$-likelihood, as it represents information about the information bit $x_k$ received directly from the channel. The channel-likelihoods are calculated as follows. Recall that the input labels $(x_i)$ on the trellis for the turbo decoder in the Wyner-Ziv codec correspond to the components of the quantized symbols $\mathbf{q}$. A pixel $u$ is quantized to $\mathbf{q}_i = [q_i^0, q_i^1, \ldots, q_i^{M-1}]$, $i = [1, 2, \ldots, 2^M]$. Then, assuming $N$ quantized symbol are grouped together before encoding, the input to the encoder can be written as $\mathbf{x} = [q_1^0, \ldots, q_1^{M-1}, \ldots, q_N^0, \ldots, q_N^{M-1}]$, where $q_l^m$ is the $m^{\text{th}}$ bit-plane in the quantized symbol representing pixel $l$. In other words, the input to the encoder $x_k = q_l^m$ for some bit-plane $m$ and pixel $l$. Thus, in order to compute the branch metric in (7), the channel-likelihoods, $P(y_k^x|q_l^m)$, need to be computed. Since the systematic bit $x_k$ is not transmitted, there is no information for $y_k^x$. However, side-information for each pixel is available at the decoder. The authors of [1], [5] use the following approach to estimate[2] $P(y_k^x|q_l^m)$ from the SI. Let $v_l$ be the side-information corresponding to pixel $u_l$ that has been quantized into symbol $\mathbf{q}_l$. Also assume that a pixel $u$ is quantized to $\mathbf{q}_i$ if $b_{i-1} \leq u \leq b_l$. Then, the probability $P(\mathbf{q}_i|v)$ is given by

$$\begin{aligned} P(\mathbf{q}_i|v_l) &= P(b_{i-1} \leq u_l \leq b_i|v_l) && (9) \\ &= P(b_{i-1} - v_l \leq \eta \leq b_i - v_l) && (10) \\ &= F_\eta(b_i - v_l) - F_\eta(b_{i-1} - v_l), && (11) \end{aligned}$$

where (10) follows from (1), and $F_\eta(\cdot)$ is the cumulative distribution function of $\eta$. The probability $P(q_l^m = j|v_l), j \in \{0,1\}$ can then be obtained by marginalization as

$$P(q_l^m = j|v_l) = \sum_{\mathbf{q}_i : q_i^m = j} P(\mathbf{q}_i|v_l), \ j \in \{0,1\} \quad (12)$$

The probability $P(v_l|q_l^m = j)$ is then obtained using Baye's rule and used to approximate the channel-likelihood $P(y_k^x|q_l^m)$ in the computation of the branch metric in (7).

There are two approximations in this approach to computing the channel-likelihoods. First, the probabilities $P(q_l^m = j|v_l)$ and $P(q_l^n = j|v_l)$, $m \neq n$ are assumed to be independent in the marginalization in (12). But this is not true. The components of the quantized symbols $\mathbf{q}$ are in fact correlated i.e., $P(q_l^1 = 0|v_l) \neq P(q_l^1 = 0|v_l, q_l^0 = 0)$. That is, given that the MSB $q_l^0 = 0$, $P(\mathbf{q}_l|v_l)$ needs to be marginalized only over the symbols for which the MSB is 0 in order to get $P(q_l^1 = 0|v_l, q_l^0 = 0)$. Second, since there is no channel output for the systematic bits, the decoder approximates the channel likelihoods $P(y_k^x|x_k)$ with $P(v_l|q_l^m)$. In the next section, we present a hyper-trellis for the turbo decoder which avoids the marginalization in (12), and the channel-likelihoods can be computed without approximation.

## III. Turbo Decoding on a Hyper-Trellis

In order to avoid the drawbacks of the approach mentioned in the previous section, marginalizing the probabilities $P(\mathbf{q}|v_l)$ should be avoided. One obvious solution is to use a turbo encoder/decoder over an $M$*-ary* alphabet. Converting $\mathbf{q}$ to binary codewords is no longer necessary if an $M$*-ary* alphabet is used. In this case, the input to the correlation channel would be the quantized version $\mathbf{q}_i$ of pixel $v_l$, and the output would be the corresponding pixel $v_l$ in the SI frame. Thus, the channel likelihood for the correlation channel with $M$*-ary* inputs will be of the form $P(v_l|x_k = \mathbf{q}_i)$, $i \in \{1, 2, \ldots, 2^M\}$. Thus, the channel likelihoods can be calculated using (11) and Baye's rule. However, the turbo encoder now requires the ability to perform operations over a higher-dimensional field which increases the complexity. Also, the decoder complexity increases exponentially because the number of states in the trellis is $M^m$ for an $M$*-ary* alphabet, where $m$ is the memory of the RSC code (a binary turbo code only requires $2^m$ states). Though the decoder in the Wyner-Ziv application is allowed to be as complicated as required, the exponential increase in trellis complexity can be prohibitive. This is especially true when a turbo code with a reasonable large memory is used in conjunction with a quantizer having many levels.

[2]This information was obtained through a private communication with A. Aaron, one of the co-authors of [1], [5].

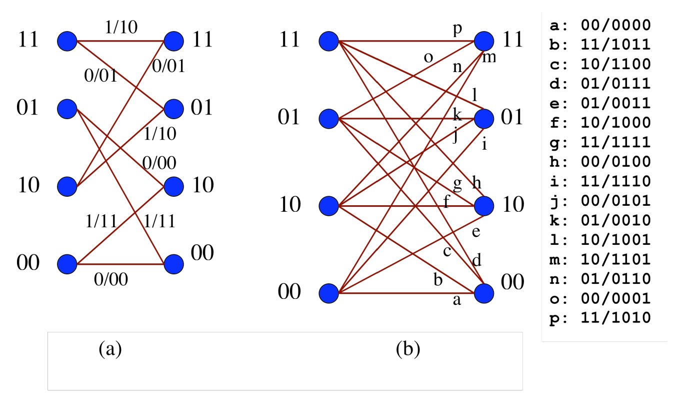

Fig. 2. $(a)$ Trellis for the $(1, 5/7)$ recursive systematic convolutional code.$(b)$ The corresponding hyper-trellis for use with 2 bit quantization (two regular trellis sections are combined to form one hyper-trellis section). Labels on the branches of the form $a/b$ imply that input of $a$ produces an output of $b$.

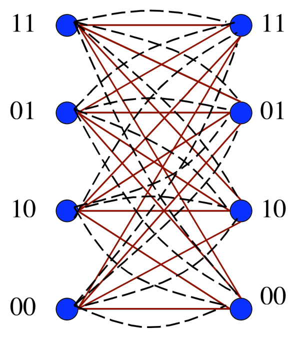

Fig. 3. A hyper-trellis section for use with 3 bit quantization (three regular trellis sections are combined to form one hyper-trellis section).

## A. Construction of the Hyper-Trellis

We avoid this increase in complexity by introducing a *hyper-trellis* structure for the turbo decoder, and implementing a modified BCJR algorithm on the hyper-trellis. The encoder does not change. The hyper-trellis is formed by merging $M$ branches of the regular trellis into one hyper-trellis section. Figure 2-$(a)$ shows the regular trellis structure for the $(1, 5/7)$ RSC code and the corresponding hyper-trellis for use in a Wyner-Ziv video codec with 2-bit quantization. Thus, two sections of the trellis section are merged to form one hyper-trellis section. The hyper-trellis is constructed as follows. Starting at time $i = 0$, we trace through all the paths through $M$ consecutive sections of the trellis. If a path starts at state $s'$ at time $i = Mk$ for some integer $k$ and ends at state $s$ at time $i + M - 1$, then all the input labels $x_i, \ldots, x_{i+M-1}$ on the branches in that path are collected to form an input label $\underline{\mathbf{X}}_i = (x_i, \ldots, x_{i+M-1})$ for the branch connecting state $s'$ and $s$ in the hyper-trellis. Similarly, the output label on the length-$M$ path in the regular trellis are grouped together to form the output label for the corresponding hyper-trellis branch. For example, let the states in the regular trellis (Fig. 2-$(a)$) be labelled $\{00, 10, 01, 11\}$. Consider the following sequence of state transitions through two consecutive sections of the trellis: $00 \rightarrow 10 \rightarrow 01$. The corresponding branch labels are $1/11$ and $1/10$. Thus, in the hyper-trellis (Fig. 2-$(b)$), there is branch connecting state $00$ and state $01$ with the label $11/1110$. This is the branch labeled $i$ in Figure 2-$(b)$. A hyper-trellis section for 3-bit quantization is shown in Figure 3. Note that the hyper-trellis has parallel transitions. Since there are $8$ possible inputs, eight branches emerge from each state, but since there are only four possible states, two branches will lead to the same next state. Thus, there are $2$ parallel branches connecting a pair of states at adjacent time intervals. The parallel transitions between a pair of states is show in Figure 3 by using a solid and a dashed line. By allowing parallel transitions, the complexity of the decoder is not increased[3] when compared to the approach used in [1], [5] and described in Section II.

Recall that the input labels $x_{i+k},\ i \in \{0, M, 2M, \ldots\}$ on the regular trellis corresponds to the $k^{\text{th}}$ component of quantized symbol $(\mathbf{q}_i)$ at time $i$, i.e, $x_{i+k} = q_i^k$. Therefore the input label on the hyper-trellis $\underline{\mathbf{X}}_i = [x_i, \ldots, x_{i+M-1}] = [q_i^0, \ldots, q_i^{M-1}] = \mathbf{q}_i$ Thus, the input labels on the branches of the hyper-trellis are the quantized symbols $\mathbf{q}$. This can be see in Figure 2-$(b)$, where the input labels correspond to the four possible quantized symbols $\{00, 01, 10, 11\}$.

## B. BCJR MAP decoding on the Hyper-Trellis

The input to the turbo encoder is $\mathbf{x} = [x_1, \ldots, x_K]$. To derive the BCJR MAP algorithm for the hyper-trellis ,we group the input bits into M-tuples and express the input as $\mathbf{X} = [\underline{\mathbf{X}}_1, \ldots, \underline{\mathbf{X}}_N]$, where $\underline{\mathbf{X}}_i = (x_{M(i-1)}, \ldots, x_{Mi-1})$, and $x_i \in \{0, 1\}$. Thus, the output of the turbo encoder can be expressed as $\underline{\mathbf{C}} = [\underline{\mathbf{C}}_1, \ldots, \underline{\mathbf{C}}_N]$, where $\underline{\mathbf{C}}_i = [\underline{\mathrm{c}}_{M(i-1)}, \ldots, \underline{\mathrm{c}}_{Mi-1}] = [x_{M(i-1)}, p_{M(i-1)}, \ldots, x_{Mi-1}, p_{Mi-1}]$. Similarly, the input to the turbo decoder $\mathbf{y} = [\underline{\mathrm{y}}_1, \ldots, \underline{\mathrm{y}}_K]$ is also grouped into M-tuples and the results vector is denoted by $\mathbf{Y} = [\underline{\mathbf{Y}}_1, \ldots, \underline{\mathbf{Y}}_N]$. Note that $\underline{\mathbf{Y}}_i = [\underline{\mathrm{y}}_{M(i-1)}, \ldots, \underline{\mathrm{y}}_{Mi-1}] = [y^x_{M(i-1)}, y^p_{M(i-1)}, \ldots, y^x_{Mi-1}, y^p_{Mi-1}]$. Since there a total $2^M$ different input labels, it is hard to define a log-likelihood ratio as in the case of binary inputs. Thus, the following developement of the BCJR algorithm for the hyper-trellis operates in the log-likelihood domain instead of the LLR domain. The log-likelihood for an input $\underline{\mathbf{X}}_{\mathbf{k}}$is defined as

$$L'(\underline{\mathbf{X}}_k = \mathbf{q}_i) = \log\left[P(\underline{\mathbf{X}}_k = \mathbf{q}_i|\mathbf{y})\right],\ i = 1, 2, \ldots, 2^M. \tag{13}$$

The decoder decides $\hat{\underline{\mathbf{X}}}_k = \mathbf{q}_i$ if $L'(\underline{\mathbf{X}}_k = \mathbf{q}_i) > L'(\underline{\mathbf{X}}_k = \mathbf{q}_j), \forall i \neq j$. Following the development in [7], the log-likelihoods can be expressed as

$$L'(\underline{\mathbf{X}}_k = \mathbf{q}_i) = \log\left(\sum_{\mathcal{X}_{\mathbf{q}_i}} \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s)\right), \tag{14}$$

where $\mathcal{X}_{\mathbf{q}_i}$ is the set of all transitions in the hyper-trellis with an input label of $\mathbf{q}_i$, and $\alpha, \beta,$ and $\gamma$ are defined in a manner

[3]Though there are more transitions per section in the hyper-trellis, the total number of hyper-trellis sections is reduced, thereby preserving the complexity.

similar to (4), (5), and (6). As in (7), the branch metric for the hyper-trellis can be expressed as

$$\gamma_k(s',s) = P(\underline{\mathbf{X}}_k = \mathbf{q}_i)P(\underline{\mathbf{Y}}_k^x|\underline{\mathbf{X}}_k = \mathbf{q}_i)P(\underline{\mathbf{Y}}_k^p|\underline{\mathbf{P}}_k), \quad (15)$$

where $\underline{\mathbf{P}}_k = [p_{M(k-1)}, \ldots, p_{Mk-1}]$ are the parity bits corresponding to the information symbol $\mathbf{q}_i$ at time $k$, and $\underline{\mathbf{Y}}_k^\star = [y_{M(k-1)}^\star, \ldots, y_{Mk-1}^\star]$, $\star = \{x, p\}$. The likelihoods for the parity symbols, $P(\underline{\mathbf{Y}}_k^p|\underline{\mathbf{P}}_k))$, can be evaluated as

$$P(\underline{\mathbf{Y}}_k^p|\underline{\mathbf{P}}_k) = \prod_{i=0}^{M-1} P(y_{M(k-1)+i}^p|p_{M(k-1)+i}), \quad (16)$$

where $P(y_i^p|p_i)$ is given in (8).

The side-information at the decoder plays the role of the received systematic symbols $\underline{\mathbf{Y}}_k^x$. Thus, the channel-likelihood can be computed as $P(\underline{\mathbf{Y}}_k^x|\underline{\mathbf{X}}_k = \mathbf{q}_i) = P(v_k|\underline{\mathbf{X}}_k = \mathbf{q}_i)$, where $v_k$ is the side-information corresponding to pixel $u_k$ (that has been quantized to $\mathbf{q}_i$). Thus, $P(\underline{\mathbf{X}}_k = \mathbf{q}_i|v_k)$ can be computed using (11) for $i = 1, \ldots, 2^M$ and then the channel-likelihoods can be obtained using Baye's rule. Note that in the hyper-trellis approach, the channel-likelihoods are computed as if an $M$*-ary* alphabet was used at the encoder. However, the likelihoods for the parity bits are computed differently.

Once the likelihoods in (14) are evaluated, it is standard procedure to extract extrinsic information to be sent to the other decoder. The extrinsic information for information symbol $\mathbf{q}_i$ is given by

$$L_e'(\underline{\mathbf{X}}_k = \mathbf{q}_i) = L'(\underline{\mathbf{X}}_k = \mathbf{q}_i) - \log\left(P(\underline{\mathbf{Y}}_k^x|\underline{\mathbf{X}}_k = \mathbf{q}_i)\right). \quad (17)$$

Note that $P(\underline{\mathbf{X}}_k = \mathbf{q}_i)$ in (15) represents the *a priori* information available to the decoder about $\underline{\mathbf{X}}_k$. In the first, iteration of the turbo decoding, $P(\underline{\mathbf{X}}_k = \mathbf{q}_i) = 2^{-M}$, $\forall i \in \{1, \ldots, 2^M\}$, and in the ensuing iterations, the extrinsic information $(L_e'(\underline{\mathbf{X}}_k = \mathbf{q}_i))$ generated by the other decoder is used as the *a priori* information.

## IV. Simulation Results

Simulation results are presented for a Wyner-Ziv codec that uses the 3GPP turbo code. The 3GPP turbo code consists of two identical RSC codes with feed-forward and feedback polynomials given by $1 + D + D^3$ and $1 + D^2 + D^3$ respectively.

The foreman sequence in QCIF format ($144 \times 176$ pixels) is used to evaluate the performance of the Wyner-Ziv codec. This is one of the standard video sequences used to evaluate the performance of video coding systems. Only the luminance values of the video sequence are used in our simulations. Quantization with four ($M = 2$) and sixteen ($M = 4$) levels is implemented. The input blocklength of the turbo code is fixed at $4800$ bits. Thus for a 2-bit quantizer, $2400$ pixels are quantized, and the corresponding binary codewords are encoded using the turbo code. For a 4-bit quantizer, $1200$ pixels are collected and quantized before encoding. In [1], [5], a feedback channel is assumed between the decoder and the transmitter. The transmitter starts by sending a small set of parity bits. If the decoder encounters a bit error rate greater than $10^{-3}$ after decoding, it requests additional parity bits. Parity bits are requested until the bit error rate is than $10^{-3}$. Because of the multiple (re-)transmissions, the number of parity bits transmitted for each frame adapts to the varying "channel" quality between the side-information and the original frame. Thus, the best possible rate (maximum compression) is achieved for each frame. This assumption of a feedback channel is not realistic, and also allowing an unlimited number of retransmissions requires a lot of buffering at the transmitter (which is undesirable). The best possible rate is the usual performance metric in source-coding problems. However, we look at the problem from a channel-coding perspective and do not worry about the best possible rate. Instead we report the rate-distortion performance for different puncturing rates. This simplifies exposition, and demonstrates the advantage of our hyper-trellis approach. It will be shown that the hyper-trellis approach performs better than the scheme in [1], [5] for all transmission rates.

We first evaluate the performance of the codec when "good" side-information is available. We generate SI for the even frames ($F_{2i}$) by using motion compensated interpolation (MCI) [8] between two consequtive odd frames ($F_{2i-1}$ and $F_{2i+1}$). The interpolation is done under the assumption of *symmetric motion vectors* (SMV) between frames (see [1], [5] for details). This technique of generating SI will be referred to as SMV-MCI. We use a search range of $16$ pixels and a block size of $16 \times 16$ pixels in the block matching algorithm (BMA) [8] used in SMV-MCI. The performance of different schemes using the SI generated by SMV-MCI is shown in Figure 4. If no parity bits are sent (zero-rate), the decoder uses the SI frames as its estimates of the even frames. This gives a peak signal-to-noise ratio (PSNR) of $28.4$ dB. This is the line labeled SI (zero bpp) in Figure 4. Note that the performance of the scheme in [1], [5] is less than $28.4$ dB for rates less than 3 bits-per-pixel (bpp). This implies that sending additional parity bits degrades performance when compared to sending no parity bits. This shows the suboptimality of the scheme used in [1], [5], because having additional information should never degrade the performance. These poor results are a consequence of approximating the channel-likelihoods (see Section II-A) in the approach in [1], [5]. The performance of the Wyner-Ziv codec using the hyper-trellis converges to zero-rate performance (MCI performance) as the transmission rate decreases. When the rate is equal to 2 bpp with $M = 2$ or 4 bpp with $M = 4$ (no compression at these rates), the performance of both the hyper-trellis scheme and original scheme coincide. This is the best possible performance achievable with the respective quantizer. With $M = 4$, there is a potential for an approximately 5 dB gain in PSNR using the hyper-trellis approach. For example, when the rate is 2 bpp (compression ratio is two), there is a $4$ dB gain in PSNR. Similarly for $M = 2$, there is a potential to increase the PSNR by $1.5$dB using the hyper-trellis approach. If a threshold PSNR of 30 dB is considered acceptable, then for $M = 4$ the hyper-trellis approach reduces the required transmission rate by over 1 bpp in comparison to the approach in [1], [5]. Note that the gain in

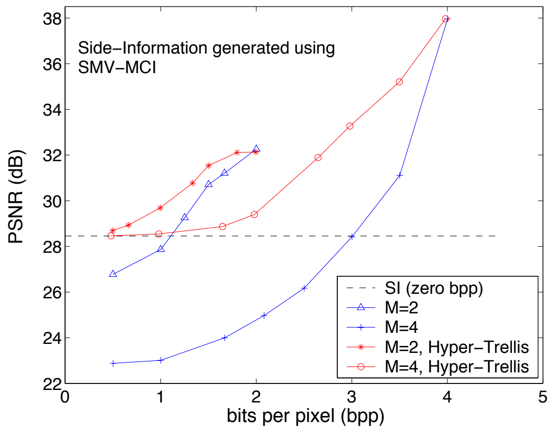

Fig. 4. Rate versus average PSNR performance for the first 400 frames of the foreman sequence. Side-information is generated using motion-compensated interpolation under the assumption of symmetric motion vectors between two consecutive odd frames of the sequence.

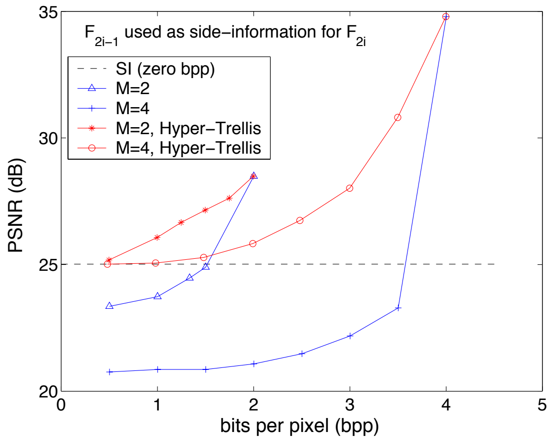

Fig. 5. Rate versus average PSNR performance for the first 400 frames of the foreman sequence. Odd frame $F_{2i-1}$ is used as side-information for $F_{2i}$.

PSNR increases with the number of quantization levels. This is because the marginalization error in (12) increases with $M$. Note that when $M = 1$, the hyper-trellis is identical to a regular trellis, and the performance of the two schemes will converge. There is no marginalzation error in this case.

The performance of different schemes when the quality of side-information is poor is given in Figure 5. The odd frame ($F_{2i-1}$) is assumed to act as SI for the even frame ($F_{2i}$). The SI is worse when compared to the previous case because MCI is not used. It is seen that the zero-rate PSNR is over 3 dB worse in this case. The various schemes behave in a manner similar to the case of using SMV-MCI to generate the side-information. At low rates (more compression), it is seen that the hyper-trellis approach provides an increase in PSNR of approximately $4-5$ dB when $M = 4$ and $1-1.5$ dB when $M = 2$.

## V. Conclusions

Wyner-Ziv coding of video is a new approach to encode a video sequence in which the source statistics are exploited in the decoder. It is shown that current implementations of the Wyner-Ziv decoder use a sub-optimal approximation in the BCJR algorithm used to estimate the *a posteriori* probabilities of the transmitted bits. It is also shown that using turbo codes operating on $M$-*ary* alphabets can resolve this sub-optimality. However, the exponential increase in decoding complexity with $M$-*ary* codes can prove prohibitive.

In this paper, a hyper-trellis structure is introduced for the Wyner-Ziv video decoder. The hyper-trellis is formed by merging consecutive sections of a regular trellis into one consolidated trellis section. The hyper-trellis structure can be considered to be a hybrid of a trellis for $M$-*ary* codes and a trellis for binary codes. The channel likelihoods in a hyper-trellis are computed in a manner analogous to the channel likelihoods for an $M$-*ary* code. However, the likelihoods for the parity bits are computed in a manner consistent with binary codes. By allowing parallel transitions between states, the hyper-trellis avoids the exponential increase in complexity required by $M$-*ary* codes. In fact, the complexity of the Wyner-Ziv decoder remains the same irrespective of whether the hyper-trellis or regular trellis is used for turbo decoding. It is shown through simulation that the hyper-trellis approach has the potential improve the PSNR by over 5 dB for the same rate, or decrease the rate required by over one bit-per-pixel for a fixed PSNR threshold. The performance improvement offered by the hyper-trellis approach increases with the number of levels ($2^M$) used for quantization.

## References

[1] A. Aaron, S. Rane, R. Zhang, and B. Girod, “Wyner-Ziv coding for video: Applications to compression and error resilience,” in *IEEE Data IEEE Data Compression Conference, DCC-2003*, (Snowbird, UT), Mar. 2003.

[2] R. Puri and K. Ramchandran, “PRISM: a new robust video coding architecture based on distributed compression principles,” in *Allerton Conference on Communication, Control, and Computing*, (Monticello, IL), Oct. 2002.

[3] D. Slepian and J. Wolf, “Noiseless coding of correlated information sources,” *IEEE Trans. Inform. Theory*, vol. 19, pp. 471–480, July 1973.

[4] A. Wyner and J. Ziv, “The rate-distortion function for source coding with side information at the decoder,” *IEEE Trans. Inform. Theory*, vol. 22, pp. 1–10, July 1976.

[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, pp. 71–83, Jan. 2005.

[6] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rates,” *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[7] W. E. Ryan, *“Concatenated Codes and Iterative Decoding”* in *Wiley Encyclopedia of Telecommunications* (J. G. Proakis ed.). New York: Wiley and Sons, 2003.

[8] Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video processing and communications*. Prentice Hall, 1st ed., 2002.