INSERTION/DELETION CORRECTION BY USING PARALLEL-INTERCONNECTED VITERBI DECODERS

T.G. Swart, H.C. Ferreira and M.P.F. dos Santos

Dept. of Electrical and Electronic Engineering Science, University of Johannesburg, P O Box 542, Auckland Park, 2006, South Africa

Abstract: A new insertion/deletion correction scheme is presented for standard convolutional codes that makes use of multiple parallel-interconnected Viterbi decoders. Whenever an insertion or deletion error occurs, the connections between different Viterbi decoders ensure that decoding will proceed from the decoder that is in synchronization. In this way, a larger Viterbi decoder is created that can correct insertion and/or deletion errors by extending the Viterbi algorithm to encompass all parallel decoders. Further, it is shown how the performance can be improved by inverting certain bits during the encoding of the convolutional codes. This lowers the frequency of occurrence of repeating sequences, which is detrimental to synchronization when dealing with insertions/deletions.

Key words: Convolutional codes, error-correcting coding, synchronization.

1. INTRODUCTION

The majority of error correcting codes are designed to correct reversal errors, also known as additive errors, where a received bit differs from the transmitted bit, e.g. in the binary case where a transmitted 0 is received as a 1 or vice versa.

Additionally, incorrect synchronization can also cause errors to occur. In this case, insertion and/or deletion errors can be used to model synchronization loss. When an insertion occurs, a random bit is received which was not transmitted and when a deletion occurs a transmitted bit is not received at all. Any synchronization loss because of these types of errors will result in bursts of reversal errors, if the synchronization is left uncorrected.

In the field of insertion/deletion correction most work has been done for block codes. See e.g. Levenstein [1], Bours [2], Davey and MacKay [3], Helberg and Ferreira [4], Swart and Ferreira [5] and references listed in these. On the other hand, relatively little research has been done in using convolutional codes to correct insertion/deletion errors. See e.g. Mori and Imai [6], Swart and Ferreira [7] and Dos Santos *et al* [8], [9].

The new scheme will be presented, along with the notion of bit inverting sequences, which improves the performance in some cases. Several factors are evaluated in the simulation results to be presented, where the performance of the new scheme is investigated.

2. PREVIOUS WORK

In this section a short review is given of some methods to correct synchronization in convolutional codes, as well as previous schemes upon which the new scheme is based.

As mentioned earlier, a synchronization error results in a burst of reversal errors. In the Viterbi algorithm, this causes all the error metrics to have a high rate of change. Synchronization is recovered by letting the framing shift a few bits while monitoring the error metrics. As soon as the error metrics' rate of change decreases significantly, synchronization has been recovered. Any data received while the Viterbi decoder is trying to recover is lost [10].

Based on this, Dos Santos *et al* [8], [9] proposed to use multiple, parallel Viterbi decoders, with each decoder one bit out of sync with the others, in order to correct insertions/deletions. By monitoring the rate of change for the accumulated error metrics, one is able to ascertain which of the Viterbi decoders is in synchronization. The output is selected from the decoder that is considered to be in sync. Although this improves on the previous method where data were lost, this scheme did not exactly determine the position of the error. Thus synchronization correction still resulted in short bursts of reversal errors, which was then corrected by using Reed Solomon codes in a concatenated approach.

The advantage of using this method, as well as the new scheme to be presented, over others, e.g. [7], is that existing convolutional codes are used, as well as standard encoding and decoding methods. Only the decoder is slightly modified. This scheme also makes use of the Hamming distance to correct insertion/deletion errors and not the Levenshtein distance which is normally associated with these types of errors and is more complex to calculate.

The scheme presented here is similar to a watermark extraction scheme proposed by Mansour and Tewfik [11]. Watermarks are inserted into regions with a high masking threshold, but after possible processing these regions can

change, causing extraction from incorrectly identified regions. These incorrect regions can be considered similar to insertion errors.

3. PARALLEL-INTERCONNECTED VITERBI DECODERS

The new scheme makes use of multiple parallel Viterbi decoders, each one bit out of sync with the others. However, each of these decoders is also interconnected to the decoder next to it. By doing this, a larger Viterbi decoder is created that can correct insertion and/or deletion errors by extending the Viterbi algorithm to encompass all the parallel decoders.

Thus, where the previous scheme of Dos Santos *et al* made use of the rate of change in the error metrics to choose which decoder is in sync, this scheme integrates this into a modified Viterbi algorithm, which finds the most likely path through the super trellis. In Figure 1, this super trellis is illustrated using an $R = 1/3$ code with two states as a simple example.

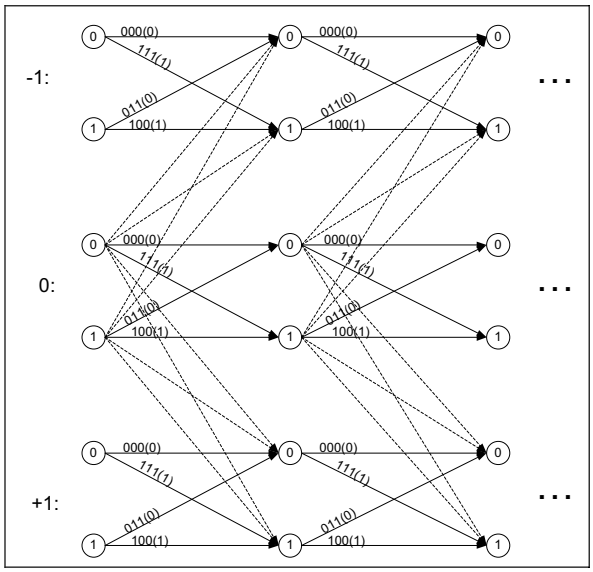

Figure 1: Representation of super trellis

For clarity, the entire decoder is referred to as the super decoder and the respective parallel decoders as the sub decoders. Each sub decoder is assigned an integer denoting its sync relative to the sub decoder that is assumed to be in sync, e.g. the sub decoder that is in sync is denoted by 0, the sub decoder lagging by one bit is denoted by -1 and the sub decoder leading by one bit is denoted by +1.

The interconnections between sub decoders (dashed paths in Figure 1) are biased so that irrespective of the received data, they will always contribute n units of distance to the error metric. This is to ensure that a path leading to another sub decoder is not mistakenly chosen unless an insertion or deletion occurred. During decoding the usual $5m$ decoding delay is used, with m the memory length, although some of the results will show that this choice of delay does not always produce the best performance.

In Figure 2, an example of decoding is shown. For simplicity only the survivor paths and the output path (in bold) are shown. Decoding has proceeded normally up to this point (Figure 2(a)). When the super decoder finds that an output indicates a switch from one sub decoder to another (Figure 2(b)), the decoder has to go back 5 time intervals (the delay used in this example), resynchronize and recalculate all the error metrics and survivor paths (Figure 2(c)). In this example it is found that the -1 sub decoder is now in sync. The -1 sub decoder becomes the 0 sub decoder and the 0 sub decoder becomes the +1 sub decoder. At this point all error metrics are set to zero and the next 5 time intervals can be recalculated from where normal decoding can resume.

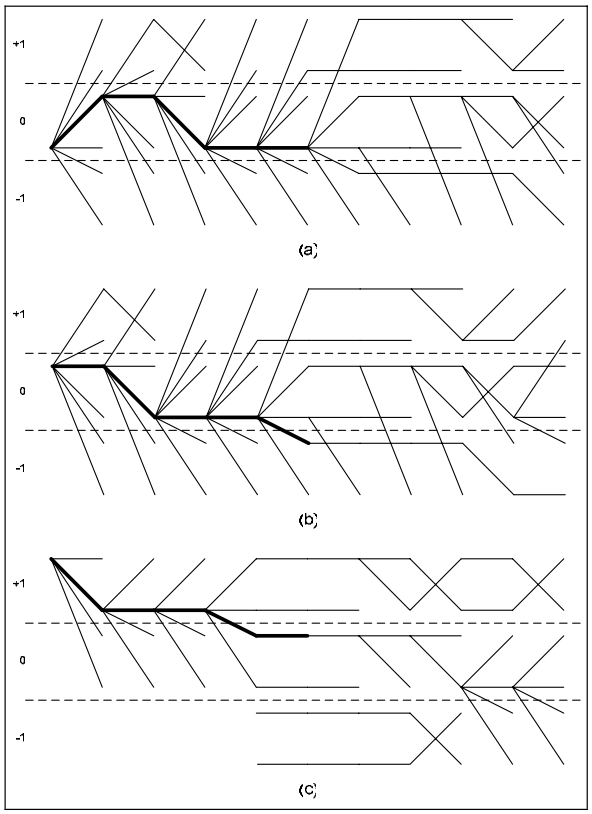

Figure 2: Trellis example of resyncing after a deletion

The insertion/deletion correction ability depends on the configuration and number of sub decoders used. When $n-1$ sub decoders are used, this scheme can correct $\lfloor n/2 \rfloor$, n odd, and $\lfloor (n-1)/2 \rfloor$, n even, consecutive deletions and/or insertions on a channel with mixed errors, i.e. insertions and deletions. When only one type of error is expected to occur and $n-1$ sub decoders are used, then $n-1$ consecutive errors can be corrected. In both cases, a guard space of at least $D \times n$ code bits with no errors is needed after the errors, where D denotes the delay used in the Viterbi decoder. In some instances this

ability can be improved on, though no guarantees can be given.

4. BIT INVERTING

Although this scheme works when using existing codes, better performance can be achieved by making slight changes to the encoder.

Considering for example the all zero path in the trellis, one would realize that an insertion/deletion in a long run of zeros would not be detected immediately. Inverting certain bits during encoding can easily eliminate this. As this is done for each word, the distance profile of the convolutional code remains unchanged, however it ensures that there is enough distance between paths that are shifted by one bit, which is exactly what the scheme relies on. This has previously been researched, see e.g. Baumert *et al* [12].

It can also be noted that the structure of the convolutional code repeats every n bits. Whenever n deletions or insertions occur, a whole word is lost or gained which the decoder will not pick up. Also, when both insertions and deletions occur, it is possible for s deletions to look like $n - s$ insertions. For example, using a code with $n = 3$, if 2 deletions occur, the decoder would detect and decode this as 1 insertion. Because convolutional code words are relatively short, this severely hampers the error correction ability of the decoder.

However, this problem can be overcome to some extent by inverting the bits in every second time interval. In every even time interval, the data is encoded as normal and in every odd time interval, the data is encoded and then the bits are inverted. Using this method, the convolutional code's structure repeats every $2n$ bits instead of every n bits, thereby increasing the error correcting capability of the decoder. The effect of these inverting bits will be illustrated in the results of the next section.

5. RESULTS

In this section, selected results will be presented to demonstrate the performance of the codes, as well as to highlight some of the key issues encountered.

The channel model used for the simulations is a symmetric reversal, insertion and deletion (RID) channel, which is a simplification of the general RID channel used by Swart [13] and similar to the channel of [3]. Similar to the Binary Symmetric Channel, this model is memory-less with statistically independent errors.

Convolutional codes with $R = 1/2$, $R = 1/3$ and $R = 2/3$ and $m = 1$ and $m = 2$ were used to generate the results presented. All simulations were performed using input data lengths of 1000 bits, unless otherwise indicated. Several thousand of these 1000 bit packets were sent over the channel. Perfect sync is assumed between each packet and in practice this can be achieved by using markers, see [14] and [15].

In Figures 3–5 the deletion error rate (DER) is shown against the bit error rate (BER). Whenever a decoding error occurred, the resulting burst of reversal errors was also included in the calculation of the bit error rate. The bit inverting sequence used for the different codes is shown in square brackets in the legend and D indicates the delay that was used. The graphs denoted as "normal" indicate the performance when the convolutional code is used without synchronization correction.

In Figure 3, the performance of the $R = 1/3$ convolutional codes is shown. A delay of $D = 5$ is used for the $m = 1$ codes and a delay of $D = 10$ is used for the $m = 2$ codes. One can see that the codes with the [00 0 00 1] inverting bits perform much better that the others at high deletion error rates. However, for very low deletion error rates the other codes' performance increases significantly and one sees that the $m = 2$ codes' performance start to coincide. A possible explanation for this will be provided shortly.

Figure 4 shows the performance of the $R = 1/2$ convolutional codes. For high deletion error rates the two codes are almost the same, however for lower deletion error rates the $m = 2$ code starts to perform slightly better.

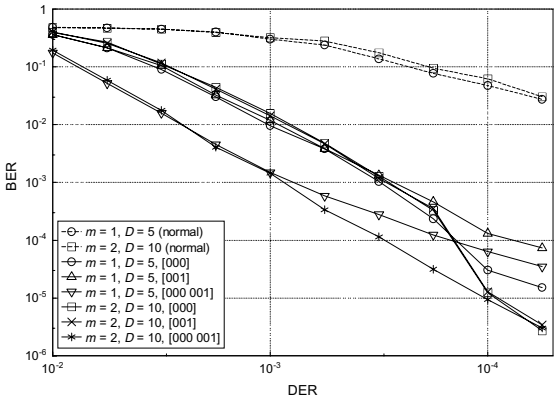

Figure 3: Bit error rate for $R = 1/3$ convolutional codes

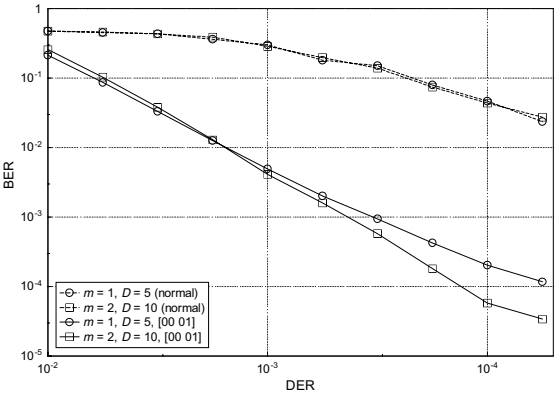

Figure 4: Bit error rate for $R = 1/2$ convolutional codes

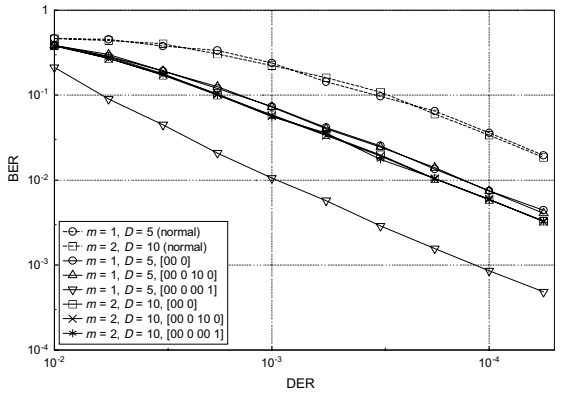

Figure 5: Bit error rate for $R = 2/3$ convolutional codes

In Figure 5, the performance of the $R = 2/3$ convolutional codes is shown. One can see that the $m = 1$ code with the [00 0 00 1] inverting sequence performs much better than the others; further research is being done to investigate this occurrence. For the other codes, one sees that the $m = 2$ codes perform slightly better than the $m = 1$ codes.

These graphs illustrate that there is an improvement in bit error rates when deletion errors are corrected. However, some of these graphs also show an error floor, where the bit error rate does not decrease significantly even though the deletion error rate is low. An investigation into this revealed that even though the deletions are corrected, a certain amount of deletions would contribute to single bit errors occurring.

In Figures 6–8 the delay length of the Viterbi decoder is shown against the bit error rate for various deletion error rates. Only specific codes were selected for these results to emphasize certain aspects.

Figure 6 shows the performance of an $R = 1/3$, $m = 1$ convolutional code with a [001] bit inverting sequence. A significant decrease in the bit error rate for $D \geq 4$ is observed, since the code is not able to build enough distance between paths for low delays. A general upward trend in the bit error rates for larger delays is observed. As mentioned in the previous section, a guard space of $D \times n$ bits is needed, thus a larger delay causes a larger guard space which leads to the decreased performance.

Figure 7 shows the similar performance of an $R = 1/3$, $m = 2$ convolutional code with a [001] bit inverting sequence. In this case however, a larger delay is necessary before the code reaches its best performance. The bit error rates again show a slow increase as the delay gets larger.

In Figure 8, the performance of an $R = 2/3$, $m = 1$ convolutional code with a [00 0 00 1] bit inverting sequence can be seen. In contrast to the previous two codes, this code shows an overall decrease in bit error rates as the delay gets larger.

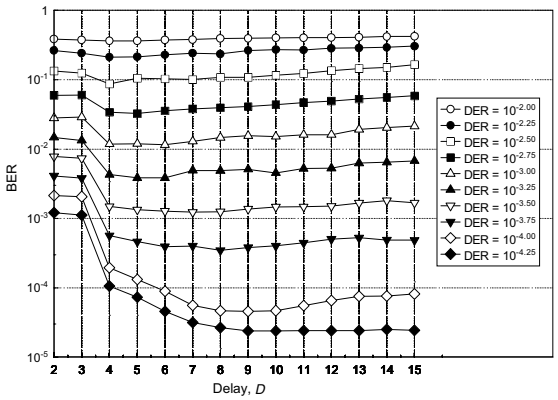

Figure 6: Bit error rate for $R = 1/3$ convolutional code with $m = 1$ and [001]

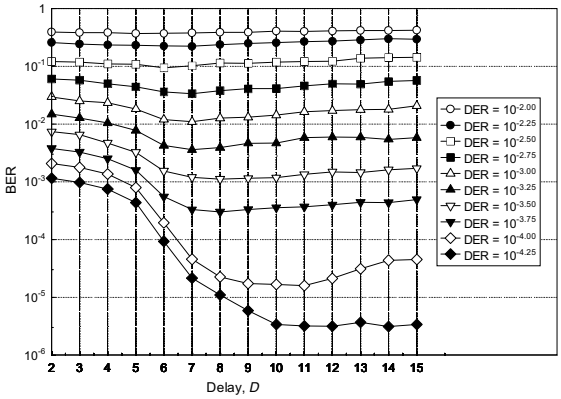

Figure 7: Bit error rate for $R = 1/3$ convolutional code with $m = 2$ and [001]

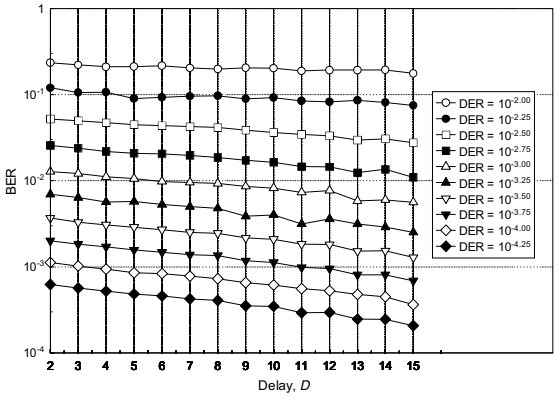

Figure 8: Bit error rate for $R = 2/3$ convolutional code with $m = 1$ and [00 0 00 1]

As was stated earlier, several thousand packets were sent during the simulations and the *synchronization loss probability* is used to show how many of these packets could not be corrected. A high synchronization loss probability will indicate that synchronization was lost in most packets, while a low synchronization loss probability will indicate that synchronization was kept or

recovered in most packets. The results show that the synchronization loss probability graphs are very similar to the bit error rates graphs. This is to be expected, since the two concepts are closely related: synchronization loss leads directly to bit errors.

In Figure 9, the synchronization loss probability of an $R = 1/3$, $m = 1$ convolutional code with no bit inverting sequence is shown. A similar trend to the bit error rate graph is observed. As the delay gets larger, the probability of losing synchronization slowly increases. This again can be attributed to the larger guard space necessary for larger delay lengths.

Figure 10 shows the synchronization loss probability of an $R = 2/3$, $m = 1$ convolutional code with no bit inverting sequence. In this case, the probability of losing synchronization decreases as the delay gets larger. This code needs a large delay length for its error correcting capabilities to be effective, even though it means that a larger guard space is necessary. It is possible that this code will also show the trend of increasing synchronization loss probability, but only at very large delay lengths.

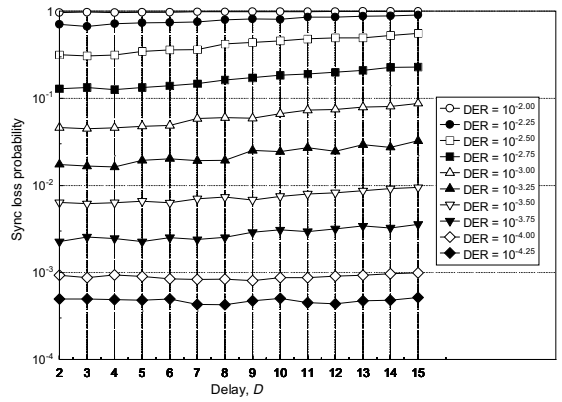

Figure 9: Synchronization loss probability for $R = 1/3$ convolutional code with $m = 1$ and [000].

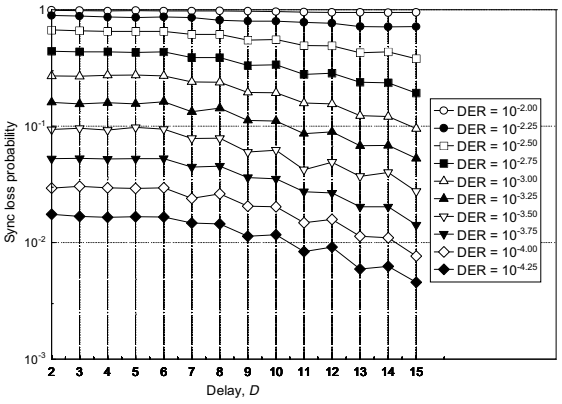

Figure 10: Synchronization loss probability for $R = 2/3$ convolutional code with $m = 1$ and [00 0].

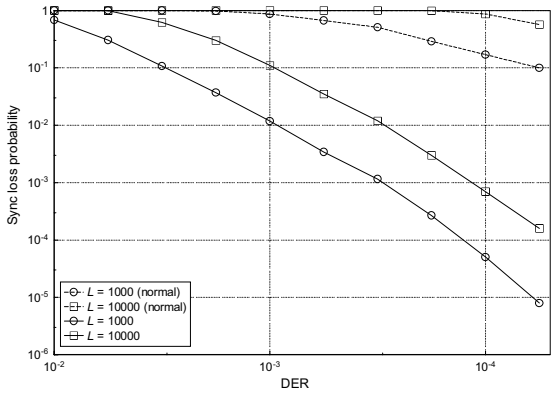

Figure 11: Synchronization loss probability for $R = 1/2$ convolutional code with $m = 1$, $D = 5$ and [00 01].

Finally, in Figure 11, we investigate the synchronization loss probability of an $R = 1/2$, $m = 1$ convolutional code with a delay length of 5 and a [00 01] bit inverting sequence. Again, the graphs denoted as "normal" indicate the performance without synchronization correction. In this case, input data lengths of 1000 bits and 10000 bits were used. Longer input lengths result in decreased performance, as a larger number of errors will occur in a packet, thereby increasing the probability of synchronization loss.

6. CONCLUSION AND FUTURE WORK

A new insertion/deletion correction scheme was presented and its performance investigated. The results showed that the scheme significantly reduces the synchronization loss probability. The results also showed that the bit error rate is lower compared to decoding without synchronization correction, although an error floor is encountered in some instances. Also, it was observed that the delay length of the Viterbi decoder plays an important role in the performance of the codes. Generally, $5m$ is an accepted delay length for normal convolutional codes, however, in this scheme this is not always effective. Longer delay lengths for normal convolutional codes increases the performance, even if very slightly, but in this scheme longer delays are detrimental to the performance. Simulation results such as the ones presented can be used to determine the optimum delay length for a code to achieve the best performance.

The performance of other codes with different rates, constraint lengths and bit inverting patterns can lead to codes with even better performance. Also, a theoretical analysis of codes to predict the performance of any convolutional code for different bit inverting patterns without having to do a simulation could be done. Finally, when a decoding error occurs in this scheme, it usually takes the form of a deletion or insertion in the output. An investigation into the possibility of concatenating two of these schemes is being done. A low rate inner convolutional code is used to correct insertions/deletions

occurring on the channel while a high rate outer convolutional code is used to correct further insertions/deletions in the output of the first super decoder. A preliminary simulation based on this concatenation has however showed no improvement, since the choice of codes seems to play an important role. More simulations with different inner and outer codes are planned.

7. REFERENCES

[1] V.I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," (in Russian) *Dokl. Akad. Nauk SSSR*, vol. 162, no. 4, pp. 845-848, 1965. English translation in *Sov. Phys.-Dokl.*, vol. 10, no. 8, pp. 707−710, February 1966.

[2] P.A.H. Bours, "Codes for correcting insertion and deletion errors," Ph.D. Thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, June 1994.

[3] M.C. Davey and D.J.C. MacKay, "Reliable communications over channels with insertions, deletions and substitutions," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 687−698, February 2001.

[4] A.S.J. Helberg and H.C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 48, no. 1, pp. 305−308, January 2002.

[5] T.G. Swart and H.C. Ferreira, "A note on double insertion/deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 49, no. 1, pp. 269−273, January 2003.

[6] T. Mori and H. Imai, "Viterbi decoding considering synchronization errors," *IEICE Transactions Fundamentals*, vol. E79-A, no. 9, pp. 1324−1329, September 1996.

[7] T.G. Swart and H.C. Ferreira, "Insertion/deletion correcting coding schemes based upon convolutional coding," *Electronics Letters*, vol. 38, no. 16, pp. 871−873, August 2002.

[8] M.P.F. dos Santos, W.A. Clarke, H.C. Ferreira and T.G. Swart, "Correction of insertions/deletions using standard convolutional codes and the Viterbi decoding algorithm," *Proceedings of the 2003 Information Theory Workshop*, Paris, France, pp. 187−190, 31 March – 4 April 2003.

[9] M.P.F. dos Santos, W.A. Clarke, H.C. Ferreira and T.G. Swart, "Correction of insertions/deletions using standard convolutional codes and the Viterbi decoding algorithm," *SAIEE Transactions*, vol. 95, no. 4, pp. 265−269, December 2004.

[10] S. Lin and D.J. Costello, Jr., Error Control Coding: Fundamentals and Applications, Prentice Hall Inc., New Jersey, First Edition, 1983.

[11] M.F. Mansour and A.H. Tewfik, "Efficient decoding of watermarking schemes in the presence of false alarms," *Proceedings of the 2001 IEEE Fourth Workshop on Multimedia and Signal Processing*, Cannes, France, pp. 523−528, 3−5 October 2001.

[12] L.D. Baumert, R.J. McEliece and H.C.A. van Tilborg, "Symbol synchronization in convolutionally coded systems," *IEEE Transactions on Information Theory*, vol. IT-25, no. 3, pp. 362−365, May 1979.

[13] T.G. Swart, "Coding and bounds for correcting insertion/deletion errors," M.Eng. Thesis, Rand Afrikaans University, Johannesburg, South Africa, March 2001.

[14] F.F. Sellers, "Bit loss and gain correction code," *IRE Transactions on Information Theory*, vol. IT-8, pp. 35−38, January 1962.

[15] H.C. Ferreira, W.A. Clarke, A.S.J. Helberg, K.A.S. Abdel-Ghaffar and A.J.H. Vinck, "Insertion/deletion correction with spectral nulls," *IEEE Transactions on Information Theory*, vol. 43, no. 2, pp. 722−732, March 1997.