Serially Concatenated Convolutional Codes with Iterative Decoding
EECS 869: Error Control Coding
Fall 2009

Complete the following tasks. You should base your implementation on the template .m files that have been distributed via e-mail. The .m files with the word `Template` in their name contain "TODOs" for you. These files should be renamed with your first name in place of the word `Template`, the TODOs should be completed, and the .m files should be sent back to me via e-mail (use the e-mail address esp@eecs.ku.edu).

1. **Specify the Trellis Table for the Differential Encoder.** In this project, we will be using the differential encoder (DE) shown in Figure 1. This encoder will serve as the "inner code" of a serially concatenated convolutional code (SCCC). The SCCC encoder is shown in Figure 2 (the symbol Π denotes an interleaver). The "outer code" for the SCCC system is the now-familiar (5,7) convolutional code (CC). We already have a trellis table for the outer code from a previous project. We need this information for the inner code as well. In the file `RunBerSimulationScccTemplate.m`, you will find a "TODO" for specifying the trellis of the DE.

2. **Implement the SCCC Decoder using the Original SISO Algorithm.** A block diagram of the SCCC decoder is shown in Figure 3. In the e-mail distribution, you already have implementations of the major blocks in this system. You also have a template function for the decoder. Your task then is to "wire" it all together properly. The data vectors that are passed back and forth between the SISO modules are in the form of log likelihood ratios (LLRs). In the case of the original SISO algorithm, the LLRs are computed in their exact form, so the scale factors K_1 and K_2 should be set to unity. The soft input $\mathbf{P}(\mathbf{u}; I)$ for both the inner and outer SISOs should be initialized to zero before the first iteration (for later iterations, $\mathbf{P}(\mathbf{u}; I)$ for the inner SISO will be non-zero). You do not need to run a BER simulation; however, the template simulation file is set up to do so. You just need to verify that the SCCC decoder works properly.

3. **Implement the SCCC Decoder using the Max-Log SISO Algorithm.** Repeat the previous task using the modified "max-log" SISO instead of the original SISO. In this case, the LLRs are computed only approximately, and the algorithm tends to "overestimate" these LLRs. Thus, there is a noticeable performance improvement if the LLRs are scaled down via K_1 and K_2. There is no analytical method for determining K_1 and K_2; however, values of 0.75 offer good performance and they also allow for a simple hardware implementation that uses shifting and adding instead of dedicated multipliers. Once again, you do not need to run a BER simulation, but you can verify that the BER performance of the max-log SISO is worse than the original SISO.

4. **Implement the SCCC Decoder using the SOVA.** Repeat the previous task using the soft output Viterbi algorithm (SOVA) instead of the SISO algorithm. You should be able to verify that the SOVA and the max-log SISO have essentially the same performance. The only difference between their performance is due to a finite traceback length within the SOVA. This will be most noticeable at high BER.

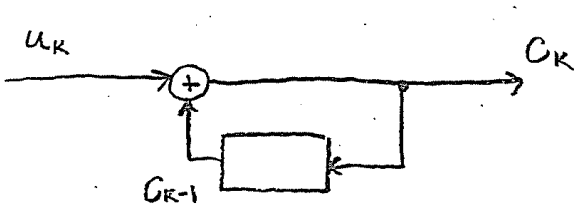

Figure 1: Differential encoder.

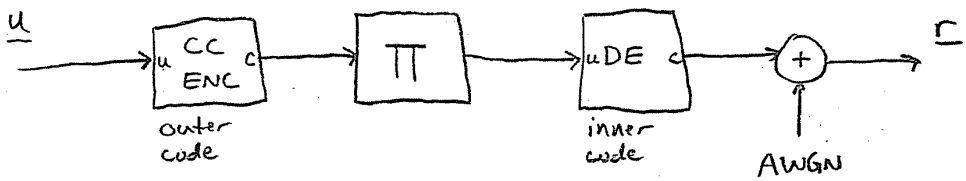

Figure 2: Serially concatenated convolutional encoder.

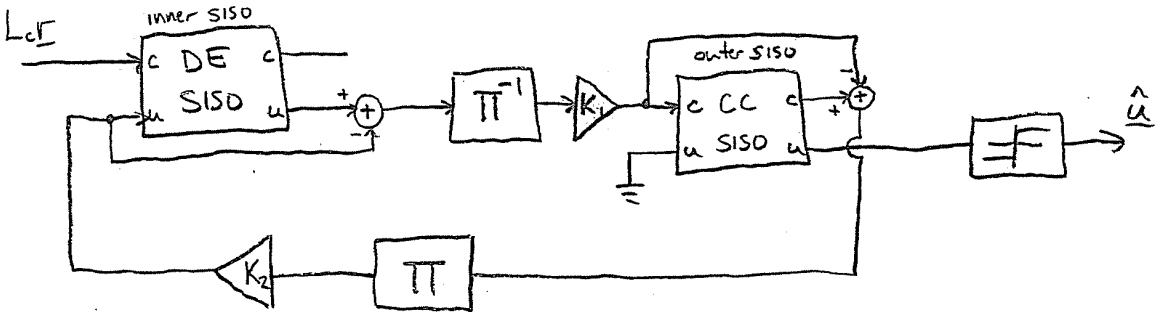

Figure 3: Serially concatenated convolutional decoder.