QPSK Modem for Satellite Communications Link

Aditya Mallick
EE21B005

***Abstract*—In this work, we design a satellite communications modem with both transmit and receive capabilities and simulate its performance in the presence of various non-idealities. In addition, we study the scope of using ML-based techniques for channel equalization. The code for the complete testbench, including individual functions, can be found at https://github.com/aditya2331/Satellite-Modem-Project/tree/master/FinalTestbench.**

I. Introduction

Receiver design for satellite communications presents a range of challenges due to the unique demands of space-based systems. The receiver must be able to effectively detect symbols in the presence of low signal-to-noise (SNR) ratio, caused by the vast distances between satellites and ground stations and atmospheric disturbances such as rain fade and ionospheric scintillation. Precise timing synchronization is critical, especially in the presence of Doppler shifts and variations caused by the motion of the satellite. Efficiently handling these issues, along with implementing robust error correction coding to mitigate the impact of bit errors, is essential to achieve the high reliability and performance required for modern satellite communication systems. The flow diagram for the complete system is detailed in Figure 2.

II. Transmitter

The job of the transmitter is to convert the required data into a form that allows it to overcome the challenges posed by the communication link and undergo accurate detection at the receiver. In this section, I go over the components that I designed to facilitate these important data transformations.

A. Error Correction Codes

Forward error correction (FEC) or channel coding is a technique used to mitigate bit errors in noisy channels. This is usually done by using an error correction code (ECC). ECCs are mainly divided into block codes and convolutional codes. In our system, we have opted to use both of these in the form of a concatenated ECC.

Concatenated codes were first regularly used for deep-space communication with the Voyager program, which launched two space probes in 1977 [1]. They eventually became a popular construction both within and outside of the space sector due to their performance advantages. Notably, they are still used today for satellite communications, such as in the DVB-S digital television broadcast standard [2]. We have implemented an outer (255, 223) RS block code and an inner (7, [171,133]) convolutional code in our system.

Personally, I was responsible for the **convolutional encoder**. Convolutional coding works by maintaining a shift register that

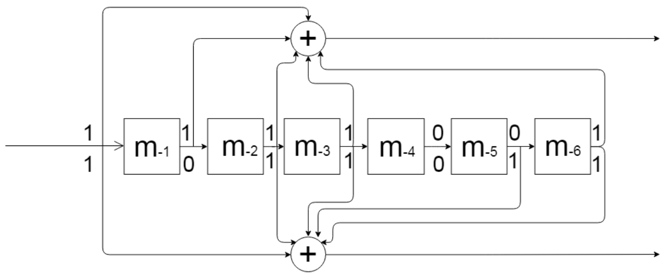

Fig. 1. Shift-register for the (7, [171, 133]) convolutional code polynomial

keeps track of previous inputs and computes the exclusive-or (modulo 2 addition) of a predetermined choice of the bits in memory to produce the output(s). Our implementation is shown in the figure above.

B. Modulation

We have opted to use QPSK modulation, a scheme popular in satellite communications. This block was also one of my responsibilities. QPSK allows us to send two bits per symbol while maintaining a low power requirement and complexity. Crucially, it is important for our assumption that the output of the convolutional code, which is two bits per input bit, can be considered to be a QPSK symbol. We have exploited this in the design of the convolutional decoder.

C. Packet Framing

Another of my responsibilities was to frame the data packets. This is done by dividing the modulated bits into a number of payload frames of a fixed size. Each of these is appended to a 26-bit preamble sequence (BPSK) that is known beforehand, forming the complete frame structure. The preamble serves two main purposes:

1) To perform correlation peak-based timing synchronization.
2) To estimate the channel coefficients for equalization.

These advantages outweigh the overhead of transmitting the preamble along with the data.

D. Upsampling and Pulse Shaping

Finally, the packets are upsampled and pulse-shaped with a square-root raised-cosine filter. The upsampling factor is determined by the design complexity of the DAC, so it will be set to a fixed number depending on the situation. We have used the SQRC pulse to perform filtering at both ends, as the raised-cosine filter satisfies the Nyquist criterion for minimum ISI.

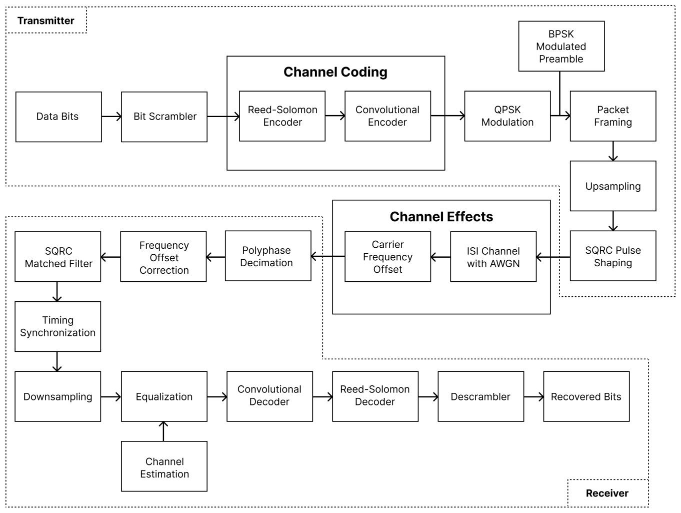

Fig. 2. System Flow

III. Channel Effects

We have modelled a distorting (ISI) channel as,

$$r(k) = \sum_{l=0}^{L-1} f_l I(k-l) + v(k)$$

where $I(k)$ is the input data to the channel having coefficients f_l. In addition, the symbols are also corrupted with AWGN having zero mean and σ_v^2 variance. Furthermore, the symbols also undergo a frequency offset, corresponding to a rotation in the QPSK constellation. All of these effects must be taken into account at the receiver for accurate data recovery.

IV. Receiver

The receiver is the most complex and important part of the system, as its design determines the overall performance of the modem. Its main purpose is to reverse all the transformations that the transmitted data has gone through and recover the original bits. This task is handled in a step-by-step manner by combining blocks that employ elegant algorithms in their implementations. Again, I will go over my contributions to the design of the receiver.

A. Timing Synchronization and Downsampling

Timing synchronization refers to aligning the receiver's sampling instants with the transmitted signal's symbol timing. It ensures that the receiver samples the incoming signal at the optimal points, where the information is most accurately captured. Timing mismatch can lead to inter-symbol interference (ISI) and poor detection performance.
Our implementation of this block utilizes the preamble sequence, which is chosen to have good autocorrelation properties, to estimate the optimal sampling point. The pseudocode for the algorithm used is provided above.

Algorithm 1 Correlation Peak-Based Timing Synchronization

```
Input: Received signal, preamble sequence, packet size, synchronization criteria
Output: Downsampled preamble and data sequences
 1: Compute correlation of received signal with preamble.
 2: Slide a window over the correlation in steps of packet size.

 3: for each window do
 4:     Update peaks, indices, and thresholds.
 5:     if synchronization achieved then
 6:         Find sampling point.
 7:         Extract preamble, data.
 8:     end if
 9:     Repeat until all packets are processed.
10: end for
11: return Extracted preambles, data.
```

Algorithm 2 Soft-Output Viterbi Equalizer

```
Input: Received signal, packet size, channel taps, preamble sequence, traceback length.
Output: Equalized soft outputs.
    Initialization: Trellis, survivors, precompute state transition mappings, set initial state to preamble.
 1: for each received symbol do
 2:     for each prior channel state do
 3:         Compute expected channel output.
 4:         Calculate branch metric as squared error.
 5:         if path metric improved for next state then
 6:             Update path metric for the next state.
 7:             Set survivor of next state as prior state.
 8:         end if
 9:     end for
10:     if sufficient symbols have been processed then
11:         Perform traceback using minimum path metric to
            equalize oldest symbol (use soft output formula).
12:     end if
13: end for
14: Perform traceback to recover final symbols.
15: return Equalized symbols.
```

B. Viterbi Equalization

We perform MLSE equalization by utilizing the Viterbi algorithm. In particular, we have implemented the soft-output viterbi algorithm (SOVA) for improved performance as well as for soft inputs to the next block, the convolutional decoder. The Viterbi equalizer provides the best theoretical performance with the downside of an exponentially increasing complexity. It also requires the channel coefficients, which are estimated by using the extracted preamble sequences from the earlier block. The pseudocode is detailed here.

C. Viterbi Decoder

My final contribution was the design of the Viterbi decoder, which has been proved to be asymptotically optimal for de-

Algorithm 3 QPSK Viterbi Decoder

```
Input: Soft equalized symbols, traceback length, generator
    polynomials (for conv code).
Output: Decoded bits.
    Initialization: Trellis, survivors, precompute state transition mappings.
1:  for each received symbol do
2:      for each future state do
3:          Compute branch metrics for all possible transitions to future state.
4:          Add these to path metric of previous state and compare.
5:          Update future state in trellis with the smallest path metric.
6:          Store the corresponding survivor state.
7:      end for
8:      if sufficient symbols have been processed then
9:          Perform traceback starting from state with minimum path metric to decode oldest symbol.
10:         Shift the trellis window forward.
11:     end if
12: end for
13: Perform traceback to recover final symbols.
14: return Decoded bits.
```

coding convolutional codes [3]. As mentioned earlier, we will take advantage of the fact that the output of the convolutional code for a single bit can be represented as a QPSK symbol (2 bits). Then, the euclidean distance between the received QPSK symbol and expected output symbol can be used as a path metric for the Viterbi algorithm, and the decoded symbols can be obtained. The pseudocode for this approach is also detailed here.

V. Simulations

To evaluate the performance of our complete system, we perform Monte-Carlo simulations to measure the average bit-error-rate (BER) against multiple levels of signal-to-noise ratio (SNR). We will study the performance of individual components of the system to confirm that they are working as intended.

- Our first experiment involves measuring the performance of the SOVA equalizer for a 4-tap ISI channel, without any channel coding. We can see that it is clearly doing its job, reducing the BER to acceptable levels at high SNR. The results are plotted in Figure 3.
- We also make sure that the convolutional decoder is working satisfactorily in the presence of no ISI channel, as is evident from the BER curve, which greatly outperforms regular hard-decision decoding and quickly hits zero errors above 2 dB. The results are plotted in Figure 4.
- Thirdly, I have repeated the same experiment as above, this time in the presence of an ISI channel, using the SOVA equalizer. Again, we can see that convolutional coding brings the the BER down to a much more acceptable level at low SNRs. In fact, this curve outperforms the previous one without any channel effects, showcasing the effectiveness of the SOVA equalizer. The results are plotted in Figure 5.

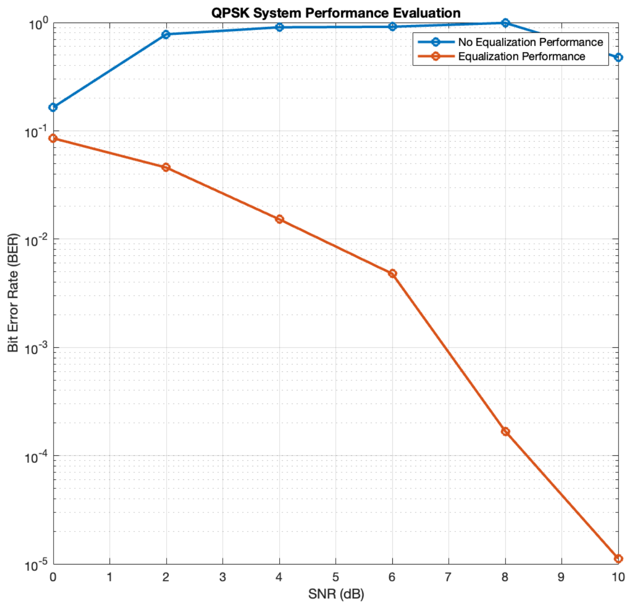

Fig. 3. Performance of SOVA Equalizer, No Coding

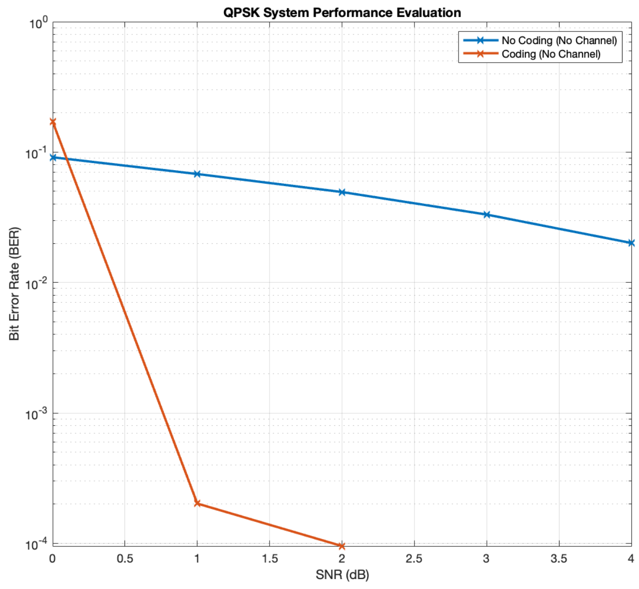

Fig. 4. Effect of Convolutional Coding - No Channel

VI. Additional Research

Independently, I decided to look into the scope of using machine learning in digital communications. My research led

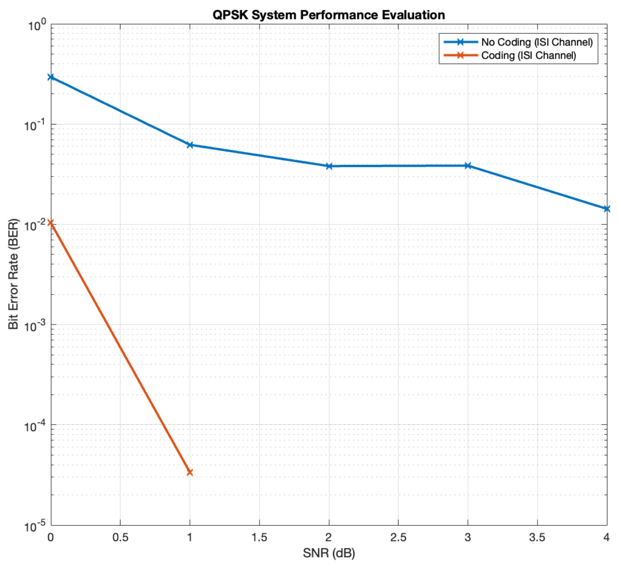

Fig. 5. Convolutional Decoding with SOVA Equalization

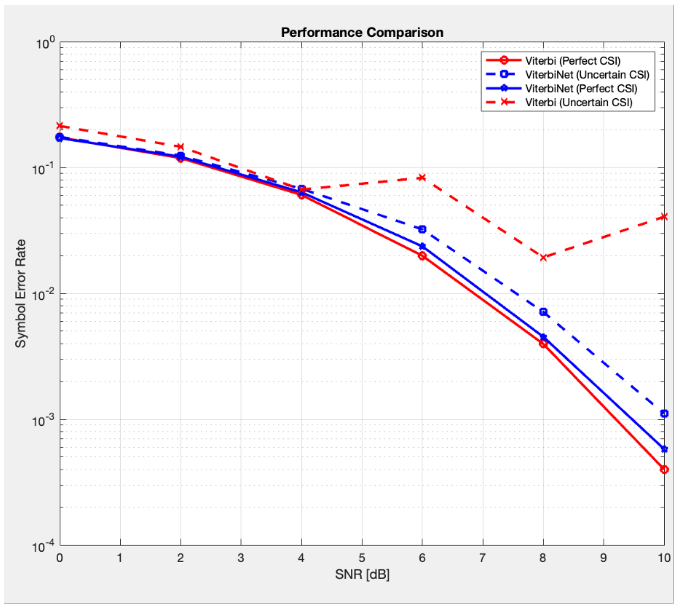

Fig. 6. ViterbiNet vs Viterbi Algorithm, varying CSI

to me experiment with ViterbiNet [4], an ML-based Viterbi Algorithm. ViterbiNet uses ML to compute the log-likelihood function for received symbols, which can be passed onto the Viterbi algorithm as a cost metric. The basic workflow of ViterbiNet is as follows:

- Use a Deep Neural Network to predict the posterior PDF $p(s|y)$, i.e the PDF of states s that the received symbol y could have originated from, assuming an LTI channel model with finite memory length.
- Use a kernel density estimation model, such as a Gaussian Mixture Model (GMM) to predict the marginal PDF of y, the output symbol.
- Use the Baye's rule and the property of the constellation symbols being equiprobable to arrive at the equation:

$$p(y|s) = \frac{p(s|y) \cdot p(y)}{m^{-l}} \tag{1}$$

  where m is the constellation size, and l is the memory length.

Once this likelihood $p(y|s)$ is computed for all received symbols over all states, it can be passed onto the Viterbi algorithm for subsequent sequence estimation.

VII. ViterbiNet Results

These experiments were done for BPSK symbols, and an ISI channel with 4 real-valued taps (for simplifying neural network architecture). Code can be found here.

- The first experiment involves testing the performance of the ViterbiNet algorithm against the Viterbi algorithm in the case of full CSI (channel state information) and in the case of uncertain CSI, where the VA is only provided a noisy estimate of the channel and VN is trained using a mixture of noisy estimates. The results show that ViterbiNet almost matches VA performance in full CSI setting, and performs much better than VA in uncertain CSI scenario. Results are in Figure 6.
- In the second experiment, we have shown that ViterbiNet, when trained on data from only a single SNR level (8 dB), can generalize well when tested across multiple SNR levels. This shows promise for training the model using a very small number of samples. In all experiments only 5000 samples were used for training. Results are in Figure 7.

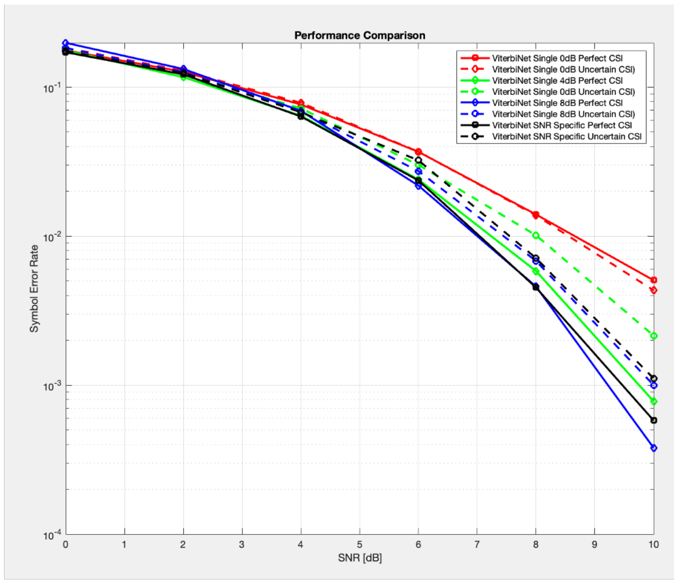

Fig. 7. ViterbiNet trained on Single vs Specific SNRs

VIII. Conclusion and Future Directions

We have addressed the main challenges faced in a satellite communications link and systematically tackled them with an extensive system consisting of multiple components. Future directions could involve using more efficient algorithms than the Viterbi algorithm and/or better performing channel codes.

In addition, our results show that ML-based methods show promise in having a role to play in the future of communications.

References

[1] K. Andrews et al., The Development of Turbo and LDPC Codes for Deep-Space Applications, Proceedings of the IEEE, Vol. 95, No. 11, Nov. 2007.

[2] Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services, ETSI EN 300 421, V1.1.2, August 1997.

[3] A. J. Viterbi, ”Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”, IEEE Trans. Inform. Theory, vol. IT-13, pp. 260-269, Apr. 1967.

[4] Shlezinger et Al., “ViterbiNet: A deep learning based Viterbi algorithm for symbol detection,” IEEE Trans. Wireless Commun., vol. 19, no. 5, pp. 3319–3331, 2020.