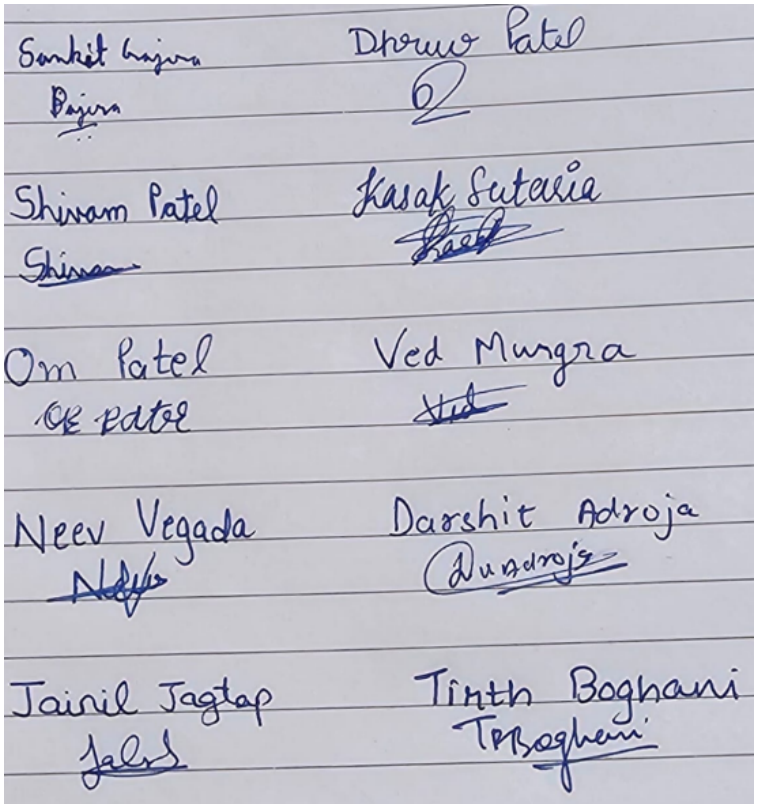
Group Number: 3

Convolution code Analysis Report

We declare that

- The work that we are presenting is our own work.
- We have not copied the work (the code, the results, etc.) that someone else has done.
- Concepts, understanding, and insights we will be describing are our own.
- We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences.



Contents

1 Transfer Function of Convolutional Code

This section presents an analysis of the transfer function of a convolutional code. We start by studying the state diagram, derive the related system equations, and finally compute the transfer function using Mason's Gain Formula. This kind of analysis is essential to understand the performance characteristics and error correction capability of convolutional codes.

1.1 State Diagram and Transfer Function Representation

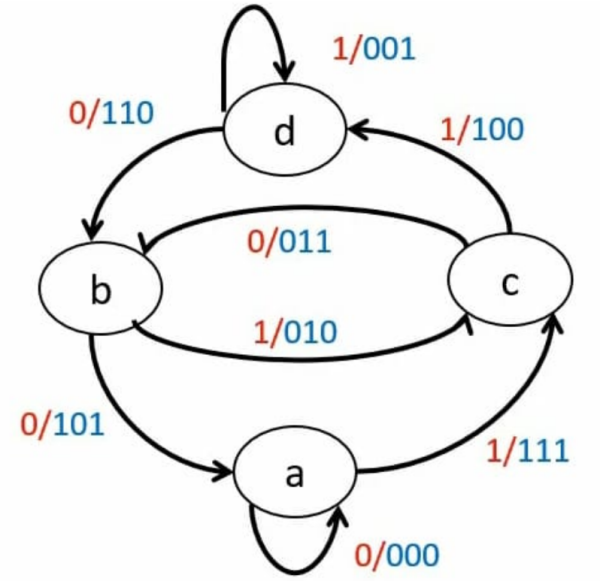


Figure 1: (a) State Diagram

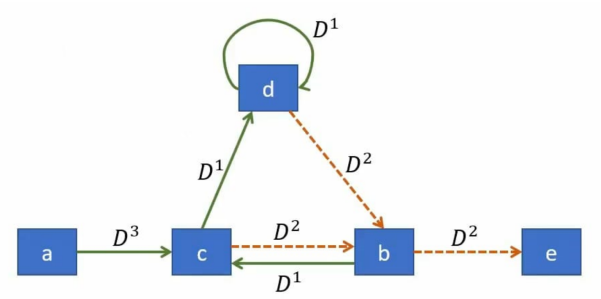


Figure 2: (b) Transfer Function Representation

The state diagram represents the transitions between encoder states for given input bits, with outputs shown in binary. The corresponding transfer function representation simplifies these transitions using weighted edges where the power of D indicates the Hamming weight (number of 1s) of the output sequence.

1.2 Transfer Function Equations

From the graph, the system equations are:

$$X_c = D^3 X_a + D X_b$$
$$X_b = D^2 X_c + D^2 X_d$$
$$X_d = D X_c + D X_d$$
$$X_e = D^2 X_b$$

The transfer function is defined as:

$$T(D) = \frac{X_e}{X_a}$$

1.3 Mason's Gain Formula for Transfer Function

To determine $T(D)$, we apply Mason's Gain Formula:

$$T(D) = \sum_{k=0}^{N} \frac{P_k \Delta_k}{\Delta}$$

Where:

- P_k is the gain of the k-th forward path,
- Δ is calculated from loop gains using:

  $\Delta = 1 - (\text{sum of individual loop gains}) + (\text{gain product of non-touching loops}) - \cdots$
- Δ_k excludes loops touching the k-th forward path.

1.4 Example Calculation

For a specific example where the following paths are considered:

Forward Path	P_k (Path Gain)	Δ_k
acbe	D^7	$1 - D$
acdbe	D^8	1

Table 1: Forward paths and their corresponding gains

The loop gain is:

$$\Delta = 1 - (D + D^3 + D^4) + D^4$$

So the transfer function becomes:

$$T(D) = \frac{1}{\Delta} \left(\frac{D^7}{1 - D} + D^8 \right)$$

After simplification:

$$T(D) = \frac{D^7}{1 - D - D^3 + D^4}$$

1.5 Free Distance and Error Correction

In this example, the minimum exponent of D in the numerator of $T(D)$ is 7. Therefore, the free distance is:

$$d_{\text{free}} = 7$$

It is important to note that this value of d_{free} is based on the specific paths chosen in this example and may vary for different encoder configurations.

A convolutional code can correct up to t errors if:

$$d_{\text{free}} > 2t$$

This relation helps in designing codes based on required error performance.

2 Hard Decision Decoding Analysis for Convolutional Codes

In this section, we study how well hard decision decoding works for convolutional codes when used over a Binary Symmetric Channel (BSC). To keep things simple, we assume that the all-zero codeword is transmitted. This is a common approach and does not affect the final result.

2.1 First Event Errors

An error occurs when the decoder picks a wrong path (instead of the correct one) for the first time. This is called a *first event error*. Let d be the number of differing bits (Hamming weight) between the correct and incorrect paths. The probability of this happening depends on whether d is even or odd.

Let p be the probability of a bit flip in the BSC. Then, the probability of first event error $P_2(D)$ is:

$$P_2(D) = \begin{cases} \sum_{k=(d+1)/2}^{d} \binom{d}{k} p^k (1-p)^{d-k} & \text{if } d \text{ is odd} \\ \sum_{k=(d+1)/2}^{d} \binom{d}{k} p^k (1-p)^{d-k} + \frac{1}{2}\binom{d}{d/2} p^{d/2}(1-p)^{d/2} & \text{if } d \text{ is even} \end{cases}$$

2.2 Error Probability over All Paths

There can be many incorrect paths, each contributing to the total error. Let a_d be the number of incorrect paths of Hamming weight d. The probability of error is upper-bounded by:

$$P_e < \sum_{d=d_{\text{free}}}^{\infty} a_d \cdot P_2(D)$$

2.3 Simplified Upper Bound Using $T(D)$

To simplify this, we use the bound:

$$P_2(D) < [4p(1-p)]^{d/2}$$

So the error probability becomes:

$$P_e < \sum_{d=d_{\text{free}}}^{\infty} a_d \cdot [4p(1-p)]^{d/2}$$

This can also be written using the transfer function $T(D)$:

$$P_e < T(D)\big|_{D=\sqrt{4p(1-p)}}$$

2.4 Bit Error Probability using Transfer Function

To find how many bits are decoded wrongly, not just whether an error occurred, we use the bit error probability P_b. For each wrong path, we multiply the pairwise error probability $P_2(d)$ by β_d, the number of incorrect information bits in that path:

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d P_2(d)$$

To make this calculation easier, we use the transfer function $T(D, N)$, which tracks both path weight and bit errors. Taking the derivative with respect to N gives us:

$$P_b < \left. \frac{dT(D,N)}{dN} \right|_{N=1,\, D=\sqrt{4p(1-p)}}$$

This gives an estimate of how often a bit will be decoded incorrectly, helping evaluate the code's performance.

3 Soft Decision Decoding for Convolutional Codes

Soft decision decoding provides a more accurate approach than hard decision decoding by considering the actual values received from the channel, rather than just converting them into 0s and 1s. This is especially useful when the channel introduces noise, as it helps the decoder make better decisions using signal reliability.

3.1 Channel Model and Received Signal

In typical systems using Binary Phase Shift Keying (BPSK), the bit 0 is mapped to $+1$ and bit 1 to -1. When a signal is transmitted over a noisy channel, it gets distorted by additive white Gaussian noise (AWGN). The received value becomes:

$$r = \sqrt{E_c}(2c - 1) + n$$

Here, E_c is the energy per coded bit, c is the transmitted bit, and n is a noise sample from a Gaussian distribution with mean 0 and variance $\frac{N_0}{2}$. The values of r are real numbers, and their deviation from ± 1 reflects how confident we are about the transmitted bit.

3.2 Metric for Decoding

The goal of the decoder is to find the code sequence that is most likely to have produced the received sequence. Instead of just counting bit mismatches (as in hard decoding), soft decision decoding uses a distance metric based on how far each received value is from the expected signal.

This distance is calculated using the Euclidean distance between the received signal and the possible transmitted sequences:

$$\text{Distance} = \sum_j \sum_m \left(r_{jm} - \sqrt{E_c}(2c_{jm} - 1) \right)^2$$

The decoder chooses the path through the trellis that has the smallest total distance, indicating the best match to the received signal.

3.3 Soft Viterbi Algorithm

The Viterbi algorithm is adapted to handle these real-valued inputs. For each state and each possible bit input, the algorithm:

- Calculates the Euclidean distance between the received segment and the expected output.
- Adds this distance to the total cost of reaching the previous state.
- Keeps track of the path with the minimum cost (called the survivor).

This process is repeated across the trellis, and once all data is processed, the best path is traced back to decode the input bits.

3.4 Pairwise Error Probability

To analyze performance, we assume the all-zero codeword is sent. If another path differs from it in d bits, the probability that this incorrect path is chosen is:

$$P_2(d) = Q\left(\sqrt{2R_c d\gamma_b}\right)$$

In this expression:

- R_c is the code rate,
- $\gamma_b = \frac{E_b}{N_0}$ is the signal-to-noise ratio per bit,
- $Q(x)$ gives the probability that a standard normal variable exceeds x.

3.5 Overall Error Probability

Many paths can cause decoding errors, so we sum the probabilities over all of them. Let a_d be the number of paths with Hamming distance d from the correct path. Then the probability of decoding error is:

$$P_e < \sum_{d=d_{\text{free}}}^{\infty} a_d \cdot Q\left(\sqrt{2R_c d\gamma_b}\right)$$

3.6 Exponential Bound

For further simplification, we can bound the Q-function with an exponential:

$$Q(x) \leq e^{-\frac{x^2}{2}}$$

This gives:

$$P_e < \sum_{d=d_{\text{free}}}^{\infty} a_d \cdot D^d, \quad \text{where } D = e^{-R_c\gamma_b}$$

3.7 Bit Error Rate Estimate

In practice, it's important to know how many individual bits are decoded incorrectly. To find this, we weigh each path's error probability by the number of wrong information bits, denoted β_d:

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d \cdot Q\left(\sqrt{2R_c d\gamma_b}\right)$$

Alternatively, we can use the transfer function $T(D, N)$, which tracks both path distances and associated bit errors. Taking a partial derivative with respect to N gives:

$$P_b < \left. \frac{\partial T(D,N)}{\partial N} \right|_{N=1,\ D=e^{-R_c \gamma_b}}$$

This method helps estimate how many bits are likely to be in error due to various paths in the trellis.

4 Performance Evaluation of SDD vs HDD

4.1 Simulation-Based Comparison

This section compares two decoding methods for convolutional codes transmitted over an AWGN channel with BPSK modulation:

- **Hard Decision Decoding (HDD)**: Reduces received signals to binary values (0 or 1), potentially discarding useful amplitude information.
- **Soft Decision Decoding (SDD)**: Retains amplitude details, giving the decoder more information to work with. This often leads to better decoding performance.

4.2 BER Trends Across Different Codes

To analyze performance, we simulated various convolutional codes and plotted their Bit Error Rate (BER) against E_b/N_0. Below are results comparing SDD and HDD under different code configurations.

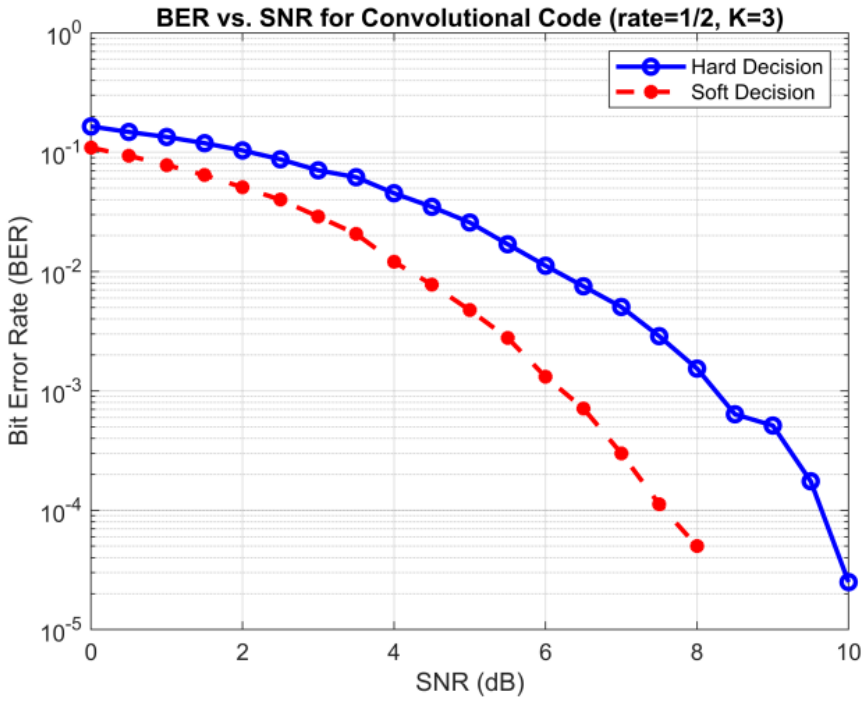


Figure 3: BER vs E_b/N_0 for convolutional code with rate $r = 1/2$ and constraint length $K = 3$. SDD shows a 2.5 dB advantage over HDD.

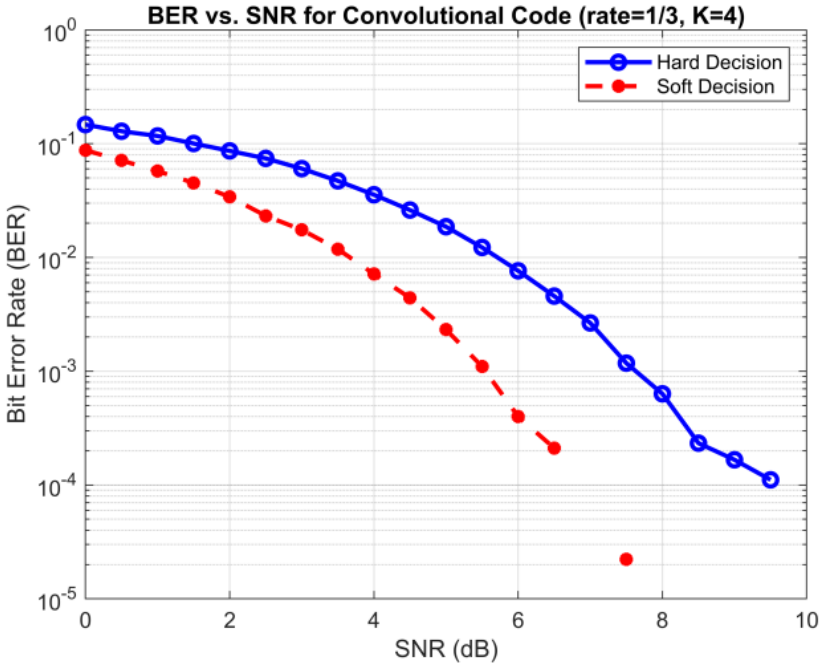


Figure 4: BER vs E_b/N_0 for code with rate $r = 1/3$ and constraint length $K = 4$. SDD achieves BER $= 10^{-2}$ approximately 2 dB earlier than HDD.

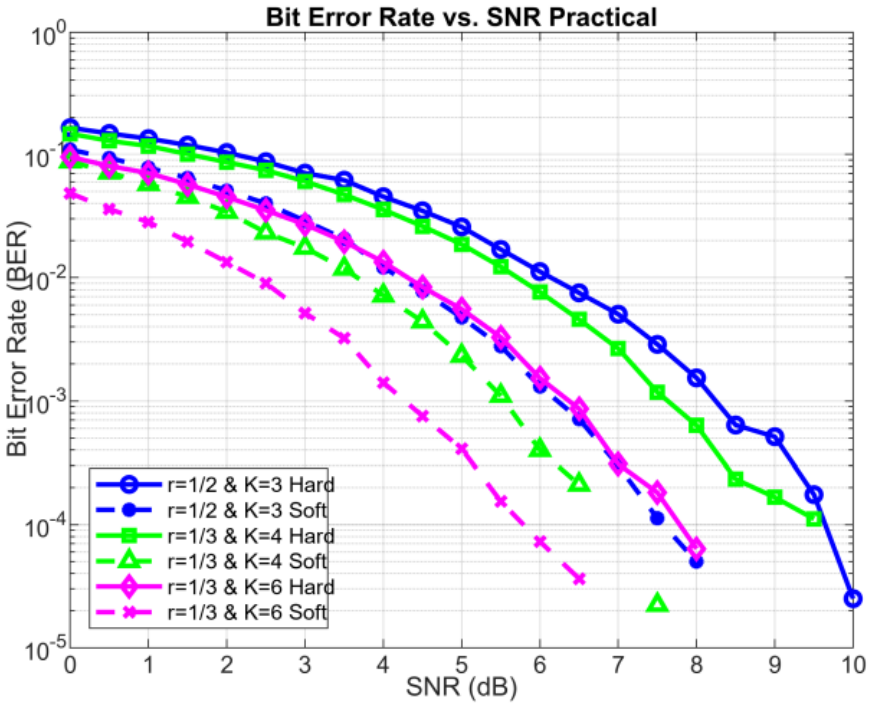


Figure 5: BER comparison across three codes: $\{1/2, 3\}$, $\{1/3, 4\}$, and $\{1/3, 6\}$. SDD consistently outperforms HDD in all cases.

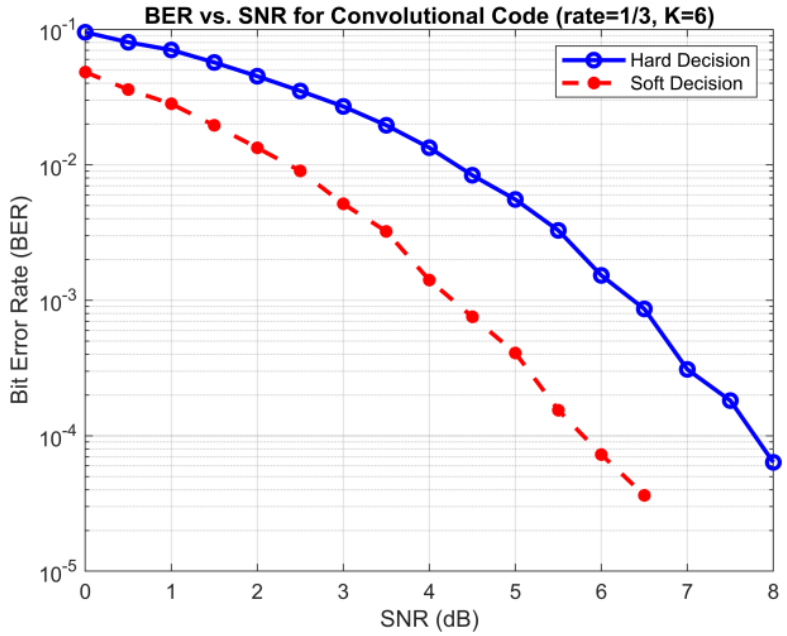


Figure 6: BER vs E_b/N_0 for high-memory code ($r = 1/3$, $K = 6$). SDD reaches BER $= 10^{-4}$ at 6 dB, while HDD requires over 9 dB.

4.3 Summary of Findings

- SDD consistently outperforms HDD, offering a signal-to-noise ratio (SNR) gain of about 2–3 dB.
- This performance gap is due to SDD's ability to use the additional amplitude information in the received signal.
- Although HDD is easier to implement, SDD is preferred in systems where achieving low BER is essential.

References

[1] J. G. Proakis, *Digital Communications*, 4th ed., McGraw-Hill, 1995.