CIE 425: Information Theory and Coding Wed., Dec. 30, 2020

Project Part 2 Report

By: Alaa Alajmy (201700095),
Hassan Youssef (201601850),
Moemen Gaafar (201700873)

Contents:

I. Introduction

Convolutional coding is a type of error-correcting code that generates parity symbols via the sliding application of a boolean polynomial function to a data stream. The sliding application represents the 'convolution' of the encoder over the data, which gives rise to the term 'convolutional coding'. The sliding nature of the convolutional codes facilitates trellis decoding using a time-invariant trellis. Time invariant trellis decoding allows convolutional codes to be maximum-likelihood soft-decision decoded with reasonable complexity.

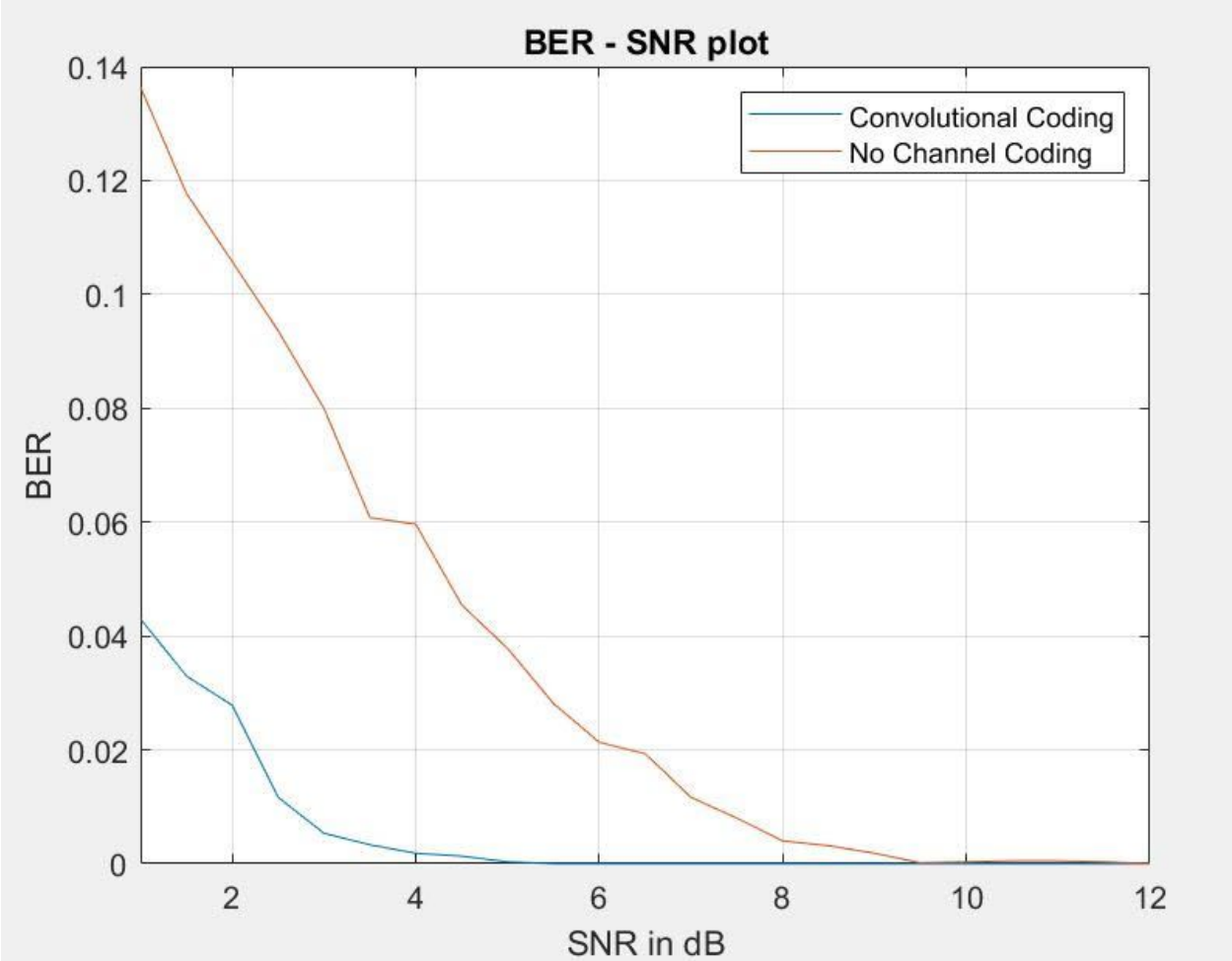
II. Methods

At first we perform huffman encoding using part 1 functions.
Part 2 contains the following functions:

a. **convEncode:**
   Input: an array of huffman encoded characters .
   Output: an array convolutionally encoded.
   This function operates convolutional encoding with k=3 and Generator polynomials: 111, 011, 101.
b. **viterbi:**
   Input: the array of convolutionally encoded sequences.
   Output: the array of decoded sequences.
   This function applies the Viterbi algorithm to decode the convolutionally encoded sequences. This function uses a smaller function which is ” getcost” that calculates the path cost of branches.
c. **Signalize:**
   Input: array of binary sequences.
   Output: array of -1,1.
   This function converts binary input to an array of -1 and 1 volts to be sent in the channel after being encoded.
d. **Binarize:**
   Input: array of -1,1..
   Output: array of binary sequences.
   This function converts the received signal of 1,-1 volts to binary signal.

III. Results

Our implementation successfully encoded and decoded the test file provided with the project statement and it returned this plot:



The plot shows how our channel coding system reduces BER significantly.

IV. Code

Our main script and our created functions are attached in the following pages.

Convolutinal Coding

Contents

Convolutional Encoder

```
function s = convEncode(m)
% Encodes input m into output s
% k = 3, r = 1/3
% Generator polynomials: 111, 011, 101
% Save length of m
n = length(m);
% Add initial zeros (2 zeros originally held in shift registers) to the start of m
% Add 2 last zeros to ensure last bits are properly corrected
m = [0, 0, m, 0, 0];
% Initialize output array of size 3*input m
s = zeros(1, n*3);
% Main loop to generate output
for i = 3:n+4
  j = i - 3;
  s(j*3 + 1)= mod(m(i-2) + m(i-1) + m(i),2); % Parity output 1
  s(j*3 + 2)= mod(m(i-1) + m(i),2); % Parity output 2
  s(j*3 + 3)= mod(m(i-2) + m(i),2); % Parity output 3
end
end
```

Function to convert binary array to array of -1's and 1's

```
function y = signalize(x)
% converts binary input to array of -1 and 1 volts
y = ones(1, length(x));
for i = 1:length(x)
    if x(i) == 0
        y(i) = -1;
    end
end
end
```

Function to convert array of -1's and 1's to binary array

```
function y = signalize(x)
% converts binary input to array of -1 and 1 volts
y = ones(1, length(x));
```

```
for i = 1:length(x)
    if x(i) == 0
        y(i) = -1;
    end
end
end
```

Function to get the branch cost from ose state to another

```
function cost = getCost(reg, input, realOutput)
% Returns the branch cost of entering bit input on registers saved in reg
% where realOutput is the 3 recieved parity bits for this "time"
output(1) = mod(input + reg(1) + reg(2), 2);
output(2) = mod(input + reg(1), 2);
output(3) = mod(input + reg(2), 2);
cost = sum(abs(output - realOutput));
end
```

Convolutional Decoder

```
function v = viterbi(s)
% Viterbi convolutional code decoder
% length of original unencoded signal
n = length(s)/3;
% Initialize costs and paths
costs = [0, 0, 0, 0];
paths = zeros(4, n*2);
tempCosts = [0, 0, 0, 0];
tempPaths = zeros(4, n*2);
% Decode first time unit
paths(:, 1:2) = [0 0; 1 0; 0 0; 1 0];
costs(1) = getCost([0, 0], 0, s(1:3));
costs(2) = getCost([0, 0], 1, s(1:3));
costs(3) = costs(1);
costs(4) = costs(2);
% If original signal has second time unit, decode second unit
if n > 1
    paths(:, 3:4) = [0 0; 0 1; 1 0; 1 1];
    costs(1) = costs(1) + getCost([0, 0], 0, s(4:6));
    costs(2) = costs(2) + getCost([1, 0], 0, s(4:6));
    costs(3) = costs(3) + getCost([0, 0], 1, s(4:6));
    costs(4) = costs(4) + getCost([1, 0], 1, s(4:6));
end
% For decode to decode till end
p = [1 3 2 4];
for k = 2:n-1
    pIndex = 1;
    pIndex2 = 1;
    realOutput = s(k*3 + 1:k*3 + 3);
    for i = 0:3
        for j = 0:1
            c = costs(i+1) + getCost(flip(de2bi(i, 2)), j, realOutput);
            if i == 0 || i == 2
                tempCosts(p(pIndex)) = c;
                tempPaths(p(pIndex),:) = paths(i+1, :);
                tempPaths(p(pIndex),k*2 + 1:k*2 + 2) = flip(de2bi((p(pIndex) - 1), 2));
                pIndex = pIndex + 1;
            else
                if c < tempCosts(p(pIndex2))
                    tempCosts(p(pIndex2)) = c;
                    tempPaths(p(pIndex2),:) = paths(i+1, :);
```

```
                    tempPaths(p(pIndex2),k*2 + 1:k*2 + 2) = flip(de2bi((p(pIndex2) - 1), 2));
                end
                pIndex2 = pIndex2 + 1;
            end
        end
    end
    costs = tempCosts;
    paths = tempPaths;
end
% Get smallest cost
[~, leastCostIndex] = min(costs);
% Get path of smallest cost
shortestPath = paths(leastCostIndex, :);
% Downsample to return decoded message
v = downsample(shortestPath, 2);
% Remove last 2 bits (2 extra zeros that were added in encoder
v = v(1: end -2);
end
```

Phase 1 function: Returns array of symbol probabilities

```
function probs = GetProbabilities(input, characters)
    charCount = zeros(1,length(characters));
    for i = 1:length(characters)
        for c = input
            if c == characters(i)
                charCount(i) = charCount(i) + 1;
            end
        end
    end
    probs = charCount ./ length(input);
end
```

Phase 1 function: Helper function for creating Huffman dictionary

```
function y = mixup(x, a)
%This function takes an array x and an array of indices a and sorts the
%elements in x by the order by indices dictated in a
n = length(x);
y = cell(1, n);
for i = 1:n
    y(i) = x(a(i));
end
end
```

Phase 1 function: Creates Huffman Dictioary

```
function codes = HuffmanCreator(symsArray, probsArray)
n = length(probsArray);
codes = cell(1, n);
temp = cell(1, n);
for i = 1:n
    temp{i} = i;
end
temp2 = [probsArray; 1:n];
for i = 1:n-1
    temp2 = (sortrows(temp2.', 1)).';
    temp = mixup(temp, temp2(2, :));
    for j = 1:length(temp{1})
        codes{temp{1}(j)} = strcat('0', codes{temp{1}(j)});
```

```
    end
    for j = 1:length(temp{2})
        codes{temp{2}(j)} = strcat('1', codes{temp{2}(j)});
    end
    temp{1} = [temp{1} temp{2}];
    temp{2} = 0;
    temp2(1, 1) = temp2(1,1) + temp2(1, 2);
    temp2(1, 2) = 2;
    temp2(2, :) = 1:n;
end
% fprintf('Symbol Code\n');
% for i = 1:n
%     fprintf('%c      %s\n', symsArray(i), codes{i});
% end
end
```

Phase 1 function: Huffman Encoder

```
function encoded = HuffmanEncoder(symsArray, codes, input)
encoded = [];
for i = input
    for j = 1:length(symsArray)
        if i == symsArray(j)
            encoded = [encoded codes{j}];
        end
    end
end
end
```

Phase 1 function: Huffman Decoder

```
function decoded = HuffmanDecoder(characters, codes, inputEncoded)
   decoded = [];
   currentCode = [];
   for i = inputEncoded
       currentCode = [currentCode i];
       for j = 1:length(codes)
           if isequal(currentCode, codes{j})
               decoded = [decoded characters(j)];
               currentCode = [];
               break;
           end
       end
   end
end
```

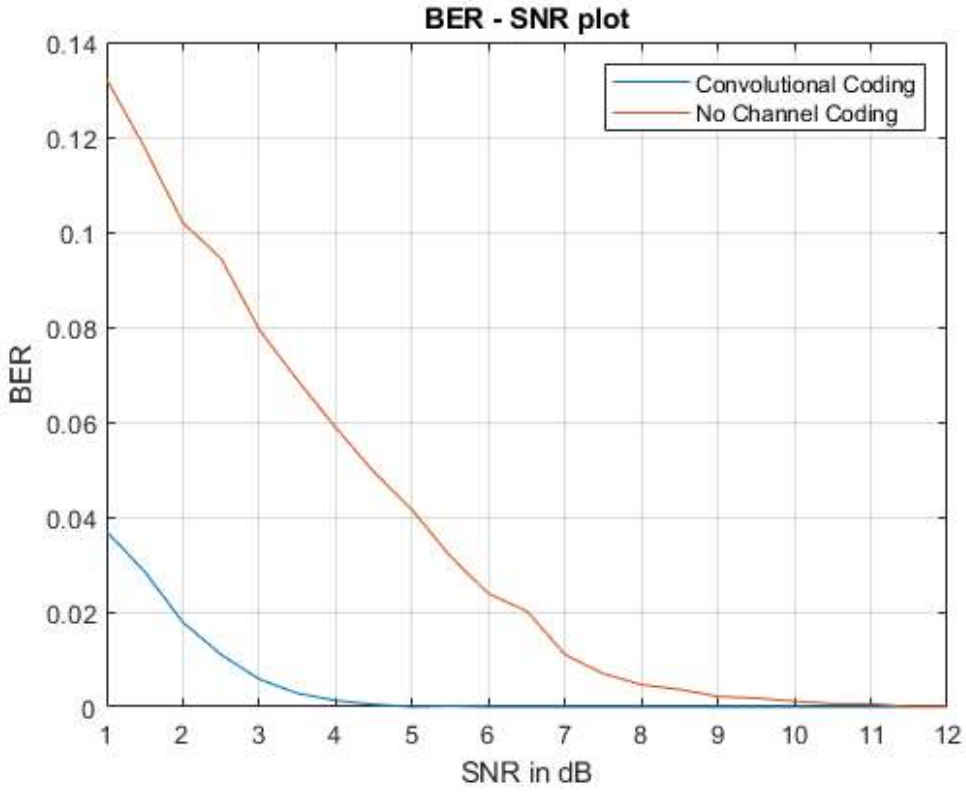
Main function

```
%Read the text file
fileID = fopen('Test_text_file.txt','r');
input = fscanf(fileID, '%c');
fclose(fileID);

fprintf("Working...\n\n");

%Calculate probability of each character
%Order: a-z ().,/-
characters = ['a':'z', ' ', '(', ')', '.', ',', '/', '-'];
probs = GetProbabilities(input, characters);
```

```
%Create Huffman code for each character
codes = HuffmanCreator(characters, probs);

%Encode test file
huffmanEncoded = HuffmanEncoder(characters, codes, input);

%Channel Encoding
huffmanEncodedArray = zeros(1, length(huffmanEncoded));
for i = 1:length(huffmanEncoded)
    huffmanEncodedArray(i) = str2double(huffmanEncoded(i));
end
bitCount = length(huffmanEncodedArray); % store the total number of bits
channelEncoded = convEncode(huffmanEncodedArray);

%Send in channel
channelEncodedS = signalize(channelEncoded); % convert sent signal to volt values of 1 and -1
%Send in channel without channel coding
noConv = signalize(huffmanEncodedArray);

BER = zeros(1, 23);
BERnoConv = zeros(1, 23);
SNR = 1:0.5:12;
for i = 1:23

  % With convolutional encoding
    %Add AWGN in channel
    channelEncodedNoisy = awgn(channelEncodedS, SNR(i), 'measured');

    %Recieve from channel
    channelEncodedNoisy = binarize(channelEncodedNoisy); % convert recieved signal to zeros and ones

    %Channel Decoding
%     t = poly2trellis(3,[7 6 5]);
%     channelDecoded = vitdec(channelEncodedNoisy, t,1 , 'trunc','hard');
%     channelDecoded = channelDecoded(1:end-2);
    channelDecoded = viterbi(channelEncodedNoisy);

    % Calculate BER
    BER(i) = sum(abs(channelDecoded - huffmanEncodedArray))/bitCount;

 % No convolutional encoding

    %Add AWGN in channel
    noConvNoisy = awgn(noConv, SNR(i), 'measured');
    %Recieve from channel
    noConvNoisy = binarize(noConvNoisy);
    % Calculate BER
    BERnoConv(i) = sum(abs(noConvNoisy - huffmanEncodedArray))/bitCount;

    % Huffman decode and write to text file for SNR = 12, 8, or 4
    if SNR(i) == 12 || SNR(i) == 8 || SNR(i) == 4

       % With convolutional coding
        %Huffman Decoding
        channelDecodedString = '';
        for j = 1:length(channelDecoded)
            channelDecodedString = strcat(channelDecodedString, num2str(channelDecoded(j)));
        end
        huffmanDecoded = HuffmanDecoder(characters, codes, channelDecodedString);

        %Write the decoded text to a separate file
```

```
        snrString = num2str(SNR(i));
        name = strcat("channelText", snrString);
        name = strcat(name, ".txt");
        fileID_Decoded = fopen(name, 'w');
        fprintf(fileID_Decoded, huffmanDecoded);
        fclose(fileID_Decoded);

       % No channel coding
        %Huffman Decoding
        noConvDecodedString = '';
        for j = 1:length(channelDecoded)
            noConvDecodedString = strcat(noConvDecodedString, num2str(noConvNoisy(j)));
        end
        huffmanDecoded = HuffmanDecoder(characters, codes, noConvDecodedString);

        %Write the decoded text to a separate file
        name = strcat("noConvText", snrString);
        name = strcat(name, ".txt");
        fileID_Decoded = fopen(name, 'w');
        fprintf(fileID_Decoded, huffmanDecoded);
        fclose(fileID_Decoded);

    end

end
%Plot BER vs SNR
plot(SNR, BER);
hold on
plot(SNR, BERnoConv);
hold off
grid on
xlim([1 12]);
legend('Convolutional Coding', 'No Channel Coding');
xlabel("SNR in dB");
ylabel("BER");
title("BER - SNR plot");
fprintf("Done!\n");
```

```
Working...

Done!
```



Published with MATLAB® R2020a