Viterbi Decoding for Convolutional Codes

Name:Sai Bhargav Dasari UnityId:sdasari

***Abstract*—The objective of this project is to analyze the performance of a 1/2 Convolutional Code with 3 memory elements using 1, 2 and 3-bit quantization of the received sequence transmitted over noisy channels using Viterbi algorithm to decode the message. By controlling the channel Signal-to-Noise ratio ,compare the BER performance of the code for the specified quantization schemes in comparison to Uncoded BER. The following report talks about the algorithmic details and own implementation of Viterbi decoding for 1-bit quantization and presents the results in comparison to Uncoded BER.**

I. Introduction

Convolutional Codes are typically represented with three parameters n,k, and K, where n and k together represent the rate of the code while K represents the number of shift registers used in the encoding part. In Convolutional Codes, the input data is not typically divided into blocks of k bits but instead a continuous stream of data is used at the encoder's input. The coded sequence of n-bits obtained after encoding not only depends on the corresponding k-bits information message but also the past information bits.In other words, these codes have memory.

This report investigates the performance of a rate(k/n) 1/2 code with Constraint length(K)= 3. The details of the encoding are provided in the following section. Section 3 talks about Viterbi Decoding and the Trellis Structure. Section 4 presents the results of the simulation and section 5 presents the code used in the simulations.

II. Convolutional Encoder

Since, the complexity of the Viterbi Algorithm is tied to the complexity of the Convolutional Code used , a rather simple non-systematic encoder mentioned in [1] of the following form $G(x) = (1 + x + x^2, 1 + x^2)$ has been used. The following picture depicts the encoding circuit [1] for the specified code:

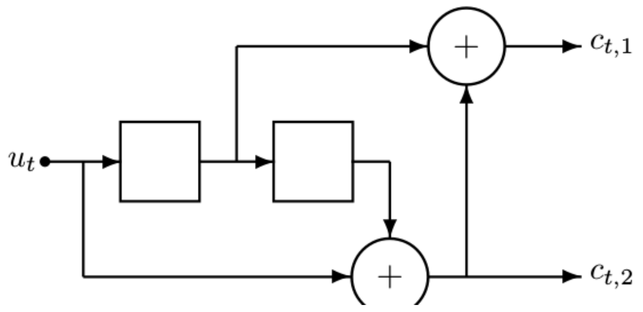

Fig. 1. Encoding Circuit with Shift Registers

As can be seen from the generating function G(x), the coded data depends on the current input bit and the two past message bits. This represents the memory part of a Convolutional Code.

After generating the codeword, it was modulated using BPSK and noise was added to it using MATLAB's *awgn* function.

III. Viterbi Decoding

Given a noisy codeword, it is inefficient in Convolutional Codes to sift through all the possible codewords and find the most likely codeword. This is because unlike block codes, convolutional codes aren't implemented for a specified block length. Usually the length of this continuous message might span several thousands of bits and as such to find the most likely codeword one might need to look at 2^{1000} combination of codewords or more and as such the memory requirements for such decoding is huge.Viterbi Decoding is a classic algorithm to recursively find the codeword closest to the received data.

The noisy codeword is demodulated and then decoded using my implementation of Viterbi Algorithm. The first step in Viterbi Decoding is to construct the trellis diagram [2] [3]. It gives information about the state transition and output information at each of the states.Given that the size of the code considered is small, instead of constructing the trellis state diagram, the state transition table and the output table are implemented independently. The state of the system is represented by the entries in the shift registers. Given that we have two shift registers, the total possible number of states is $2^2 = 4$.Also, since 1-bit quantization is considered we have two additional columns.One each for input bit=0,1. The following tables show the state transition table and output table for the encoder mentioned in section 2:

TABLE I
State Transition Table

Previous State	Input 0	Input 1
00	00	10
01	00	10
10	01	11
11	01	11

To ease the implementation, the states $(00, 01, 10, 11)$ are represented in the code as $(0, 1, 2, 3)$.The Viterbi-Decoding uses the Hamming Distance between the received codeword and the set of possible outputs at each instance to compute the error metric for each state and stores for each state it's preceding state with the least error metric. For a codeword of length L, Viterbi Decoding needs memory on the order of

TABLE II
OUTPUT TABLE

Previous State	Input 0	Input 1
00	00	11
01	11	00
10	10	01
11	01	10

$L * 2^{K-1} = 4L$ as opposed to the 2^L needed by standard decoding.

IV. RESULTS

This section discusses results of simulations done on MATLAB. Analysis of performance is done with respect to BER. BPSK is the modulation scheme utilized and the noise added is additive AWGN noise. The following plot depicts the performance plot for Coded and Uncoded bits for a message sequence of length 10^5 bits. The code used for this simulation is specified in the next section.

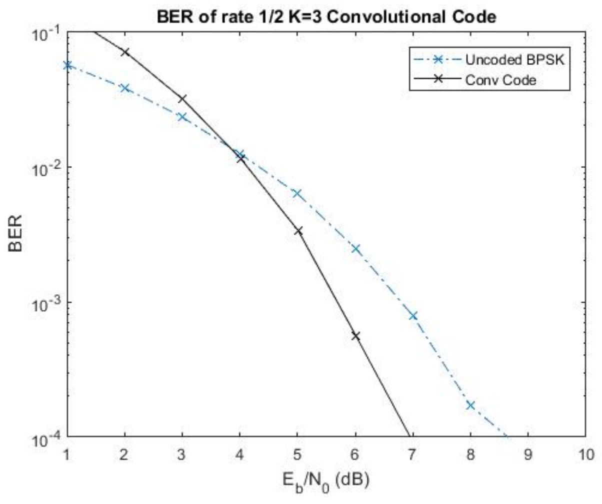

Fig. 2. Performance Comparison Plot

As can be seen from the picture for an error rate of 10^{-4} the Convolutional Code gives an improvement in performance around 1.6 dB.

V. FURTHER WORK

A fair amount of time went into writing and debugging the algorithm and as such I didn't have time to include in the report all the details regarding the Viterbi Algorithm. However, the code is heavily commented and explains each step of the algorithm quite clearly.

The goal for the second interim report is to evaluate and document the effects of 2-bit quantization of the received signal. An improvement in performance is expected because more information is represented compared to the 1-bit quantization case.

VI. REFERENCES

[1]Roth, Ron. Introduction to coding theory. Cambridge University Press, 2006.

[2]Tutorial on Convolutional Coding with Viterbi Decoding. home.netcom.com/~chip.f/viterbi/tutorial.html

[3]Shu Lin, Daniel J. Costello, Error Control Coding-Fundamentals and Applications, 2ed

```
function [predecessor_states,error_Table,optimum_path,data] =
 viterbidec(K,coded)

%Function implements the hard decision viterbi decoding
% K denotes the constraint length and coded is the demodulated binary
% sequence

%Intializing arrays essential for the state decoding process
%The State Transition table: the next state given the current state
 and
%input
transition_Table=[0 0 2;1 0 2;2 1 3;3 1 3];
%The output table:determins the output given the current state and the
%input
output_Table=[0 0 3;1 3 0;2 2 1;3 1 2];
%Error Metric Array-Track of accumulated error
error_Table=zeros(2^(K-1),1);
%Initialize the current state to be always 0
current_state=[0];
%State predecessor history-local optimum at each time instant
predecessor_states=zeros(2^(K-1),length(coded));

%Main section where magic happens
for i=1:length(coded)
    %possible states the algorithm might end up in given it's current
 state
    possible_states=transition_Table(current_state+1,2:3);
    %possible outputs the algorithm might end up in given it's current
 state
    possible_outputs=output_Table(current_state+1,2:3);
    possible_states=reshape(possible_states',1,[]);
    possible_outputs=reshape(possible_outputs',1,[]);
    %Distance between received sequence and possible outputs
    ham_distance= sum(abs(dec2bin(coded(i),2)-
dec2bin(possible_outputs,2)),2);
    %A variable to keep track if a state has aleady been updated
 during the loop
    flag=zeros(2^(K-1),1);
    %A duplicate error metric variable to keep track of error in the
 for loop
    error_Table2=nan(2^(K-1),1);
    %loop through current states
    for count=1:length(current_state)
        count2=transition_Table(current_state(count)+1,2:3);
        %update error metric for the possible states
        if(sum(flag(count2+1))==0)
            error_Table2(count2+1,1)=
 error_Table(current_state(count)+1,1)+ham_distance(2*count-1:2*count,1);
            flag(count2+1)=1;
            predecessor_states(count2+1,i)= current_state(count);
        else

 error_temp=error_Table(current_state(count)+1,1)+ham_distance(2*count-1:2*count,1
 [error_Table2(count2+1,1),ind]=min([error_Table2(count2+1,1)';error_temp']);
            change_index=find(ind==2);
            if(~isempty(ind==2))

 predecessor_states(count2(change_index)+1,i)=current_state(count);
            end


        end

    end
    error_Table2(isnan(error_Table2))=0;
    error_Table=error_Table2;
    %update current state to possible state for next iteration
    current_state=unique(sort(possible_states));
end

%Traceback
%End of block is zero since it's just flush bits
optimum_path=predecessor_states(1,length(coded));
for count3=length(coded)-1:-1:2
    path= predecessor_states(optimum_path(1)+1,count3);
    optimum_path=[path optimum_path];
end

%Decode the optimum path
current_state=0;
data=[];
for count4=1:length(optimum_path)
    if(transition_Table(current_state+1,2)==optimum_path(count4))
        data=[data 0];
    else
        data=[data 1];
    end
    current_state=optimum_path(count4);
end
%last bit is 0.(The impact of flush bits)
data=[data 0];

end
```

Published with MATLAB® R2016b

```
%main.m
%Convolutional Encoding
K=3;  %Constraint Length
t = poly2trellis(K,[7 5]);
%size of input data msg
%IT SHOULD BE EVEN
N=100000;
%generate random sequence
msg=randi([0,1],1,N);
%Flush bits-Add Two zeros at the end
msg=[msg 0 0];
code = convenc(msg,t);%Encode
transmitted=pskmod(code,2); %BPSK modulation

%AWGN Noise.Comparison with Uncoded vs Convolutional Coding
snr=-2:9;
for i=1:length(snr)
    snr(i)
    noisy_signal=awgn(transmitted,snr(i),'measured');
    demod=pskdemod(noisy_signal,2);
    [~,ber(i)]=biterr(code,demod);
    %Convert message from binary into states for input to viterbi
 decoding
    transmitted_states=bin_to_states(demod);
    %Decoding using my Viterbi Algorithm
    [~,~,~,decoded_data] = viterbidec(K,transmitted_states);
    [~,codedber(i)]=biterr(msg,decoded_data);
    ebn0(i)=snr(i)-10*log10(1/2);%Rate-1/2 convolutional code
end

%Plotting
semilogy(snr,ber,'x-.',ebn0,codedber,'xk-')
axis([1 10 0.0001 0.1])
xlabel('E_b/N_0 (dB)');
ylabel('BER');
title('BER of rate 1/2 K=3 Convolutional Code')
legend('Uncoded BPSK','Conv Code')
```

Published with MATLAB® R2016b

```
%%Auxiliary function needed for main to run
function msg2= bin_to_states(msg)
msg2=[];
for i=0:(length(msg)/2)-1
    msb=2*i+1;
    lsb=msb+1;
    msg2=[msg2 2*msg(msb)+msg(lsb)];
end
end
```

Published with MATLAB® R2016b