PROJECT REPORT

A Bandwidth Efficient Technique For Spatial Modulation

And Analysis Of Shift Encoder Techniques

Submitted for the course: Information Theory and Coding (ECE-4007)

By

16BEC0098	PRANOY DEV

Name of faculty: BUDHADITYA BHATTACHARYYA

(SENSE)

April, 2019

CERTIFICATE

This is to certify that the project work entitled "**A Bandwidth Efficient Technique For Spatial Modulation And Analysis Of Shift Encoder Techniques "** that is being submitted by Pranoy Dev for the course Information Theory and Coding (ECE4007) is a record of bonafide work done under my supervision. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted for any other CAL course.

Place: Vellore, Tamil Nadu

Date: 4th April, 2019

Signature Of Students:

Pranoy Dev

Signature Of Faculty: Prof. Budhaditya Bhattacharyya

ACKNOWLEDGEMENT

The completion of this undertaking could not have been possible without the participation and assistance of so many people whose names may not all be enumerated. Their contributions are sincerely appreciated and gratefully acknowledged. However, the group would like to express their deep appreciation and indebtedness particularly to Prof. Budhaditya Bhattacharyya for his guidance and support in our project.

Also we would like to thank the University Management and the Dean of School of Electronics and Communication for giving us an opportunity to carry out our studies at VIT University, Vellore.

To all relatives, friends and others who in one way or other shared their support, either morally, financially and physically.

We thank you.

ABSTRACT

The need for high data rate and high spectral efficiency are the key elements that drive research in future wireless communication systems. Power and bandwidth are limited resources in wireless communication systems. There has been a great improvement in system spectral efficiency with the advent of MIMO (multiple-input-multiple-output) techniques. The use of multiple antennas at the transmitter and receiver sides can cause an increase in the capacity and reliability of wireless links. But multi-antenna operation faces some challenges due to complexity and cost of the hardware owing to the requirement of inter-antenna synchronization and maintenance of multiple radio-frequency (RF) chain.

Spatial modulation (SM) is a relatively new modulation technique for multiple antenna systems which deal with these issues. Spatial modulation (SM), utilizes the multiple transmit antennas in a different way. Multiple antennas are considered as additional constellation points that are used to carry information bits. In this only one antenna will be active at a time and the active antenna index will be transmitted along with the information bits. The novel idea in this paper is to encode the antenna index by trellis coded modulation (TCM) .

There are several coding techniques such as Huffman Coding and LZW coding which can be employed to send strings from one server to other server. However it is difficult to ensure proper security for the data sent. This can be ensured by using the use of shift encoder. The shift encoder works on the principle of rotate circular shift. The encryption of the nth block of data is defined by the last k-3 bits of the (n-1)th block. Where n is the number of total blocks sent over a particular point of time and k is the code word length. This method ensures that each block has bits rotated circularly by a number ‘j’. This number ‘j’ is the decimal equivalent of the ‘binary number j’.

This encoding is helpful because even if a block of bit is attacked from outside, the illegal person will not be able to decrypt the message if he doesn’t have the previous block. Similarly even if he manages to get the prior block he doesn’t know the extent of rotation onto it as he doesn’t have the block prior to it. This continues so on.

INTRODUCTION

There is a wide increase in data rates due to emerging new technologies and it makes wireless communications a challenging field. The spectrum or bandwidth available to the service provider is often limited. The power requirements are such that the devices should use as little power as possible to conserve battery life. Thus, the designers for wireless systems face a two-part challenge, increase data rates and improve performance with little or no increase in bandwidth or power. The error correcting codes can be used to increase the bit error ratio by adding redundant bits. But the main disadvantage of that is bandwidth expansion. So for a band limited systems such as a telephone system, use of error correcting codes are difficult.

In order to use bandwidth efficiently without increasing power, a new method called **trellis coded modulation** can be used. It is a method which combines the error correction and modulation functions of the communication system, to achieve better gains in system performance. In TCM, the bandwidth expansion is not required because it uses the same symbol rate and power spectrum. The main differences are the introduction of a redundancy bit and the use of a constellation with double points.

TCM combines the function of a convolution encoder of rate R = n/(n + 1) where n is the output of convolution encoder and a M-ary signal mapper that maps M = 2^(n+1) constellation points. The main idea of TCM modulation is to partition the symbols into different sets of same size and in each set the distance between the symbols should be maximized.

In MIMO system all the antennas will be active at both the transmitter and the receiver of a wireless system. In MIMO system the complexity, power and the cost of hardware is increased. In order to combat these problems a new modulation scheme i.e. spatial modulation is used. In the normal case two dimensional modulations are used. But in the case of Spatial Modulation, it uses a third dimensional modulation. By this it means that the index of the antenna is transmitted along with the symbol. In the basic form of Spatial Moduation, the transmitter has access to all antennas, but only one out of these antennas is active and is used to transmit data in any given symbol interval. The receiver must determine which of the antennas was selected for transmission.

MATHEMATICAL MODEL

TCM Concepts:

1. Euclidean Distance : A straight line distance between any two points is called the Euclidean Distance. For a point p_1 at (x_1, y_1) and another point p_2 at (x_2,y_2), the Euclidean distance is given by the formula:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

2. Hamming Distance : It is the distance between two binary numbers. A zero distance means the numbers are the same.

The probability of error is an inverse function of the sequence length. In general form the probability of error between sequences is given by the expression :

$$p_e \sim e^{-d^2_{\min}/2\sigma^2}$$

Where d_{min} is the sequence Euclidean distance between sequences and is σ^2 the noise power.

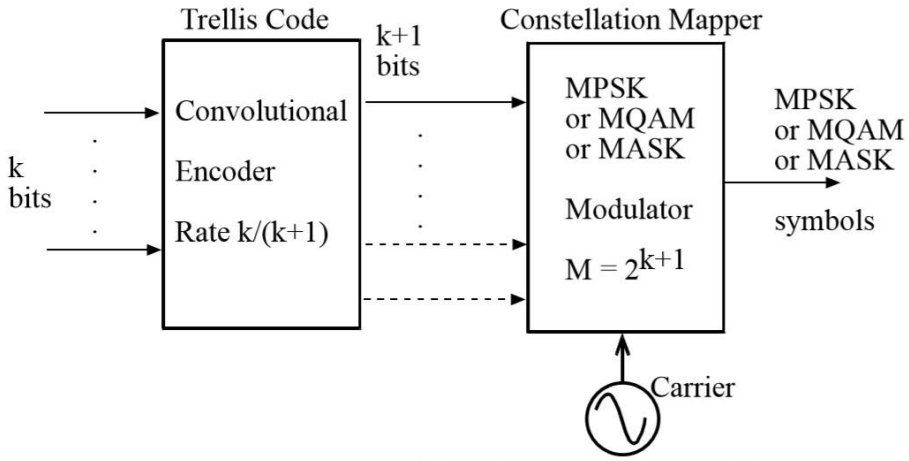
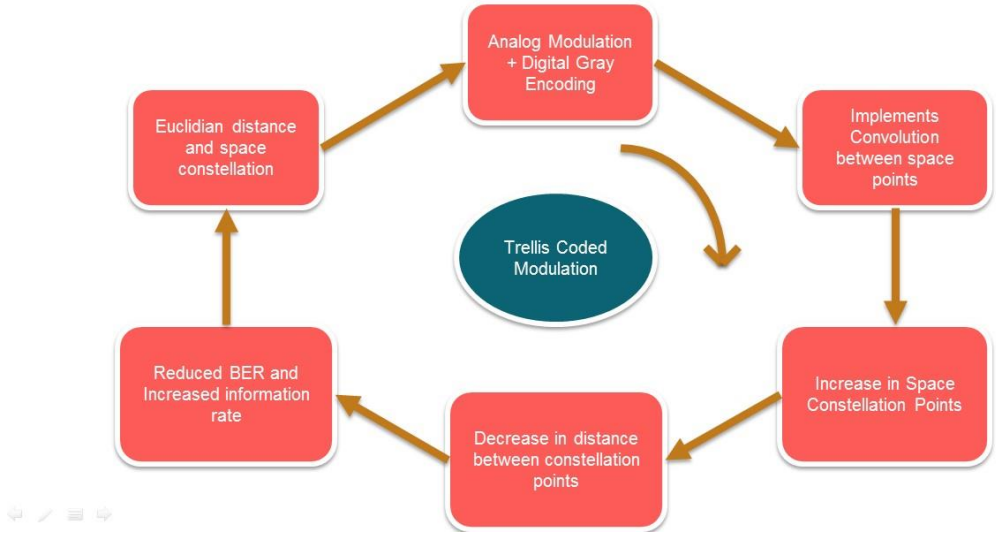


Figure 3 – A general trellis coded modulation

METHODOLOGY:

The function of TCM consists of a Trellis code and a constellation mapper. TCM combines the functions of a convolution coder of rate R = k/k+1 and a M-ary signal mapper that maps M = 2^k input points into a larger constellation of M = 2^{k+1} constellation point.

- In general the coding maps information bit to higher number of code bits.
- If the coded bits are transmitted bit by bit in order to maintain the data rate we have to increase the channel usage rate.
- More code bits means that spectral efficiency is decreased since more bandwidth is required.
- The trellis coded modulation introduces additional parity bits and does not increase the bandwidth.
- The effective throughput in the channel is maintained by enlarging the number of constellation points.
- By increasing the amount of constellation points we increase the signal set and more information can be transmitted by each signal.
- The signalling rate is not increased since each symbol contains more information.
- Since power ber bit is kept constant after increasing the constellation size the distance between the possible constellation points, symbols, is should be decreased.
- The positive coding gain is achieved if the increase of the error probability due to smaller distance between constellation points is outweighed by the coding gain of the error correction code.



Shift Encoder Encryption

- The shift encoder is a encoding scheme that can be easily encrypted and prevents loss of day while communication
- It works on the principle of usage of Rotate Right block implementation onto the transmission circuit and Rotate Left block on the receiver side.
- The encryption of any nth block of bits, happens with its dependence with the data in the (n-1)th block of data.
- For our project we concentrated on the last 3 bits of any block of the data.

- The last 3 bits of the (n-1)th block of the data will give a decimal equivalent.
- This decimal equivalent value is used to shift the nth block by exactly the number. This process continues till all the bits are transmitted.
- This procedure is safe as even if a block of code is captured by someone illegally. That person cannot decrypt the message without knowing which block of data was transmitted before it and which was transmitted even before it. So the decryption needs all the blocks at once.

WORKING

TCM-

The trellis coded modulation has different data points spatially arranged. This spatial arrangement is done according the message point's constellation orientation. Suppose if a particular constellation has n points. The hamming distance will be some value. As we know that the Bit Error Rate (BER) is inversely proportional to the minimum distance, we will exploit this factor to reduce the bit error rate. The overall efficiency of any system is defined by the number of error segments it generates over a particular transmission operation. The technique TCM maps the n points to generate new 'k' points by using the convolutional mapper. The generator matrix for the convolution helps in generation of new spatial points that can be mapped. Since the TCM uses the overall hamming distance in the least possible path. As the number of spatial points are increased the hamming distance due to the newly created spatial points will increase. This will result in overall increase in the minimum path distance taken by the Trellis Tree to encode the input. Since

$$Pe \propto \left(\frac{1}{e^{dmin^2}}\right)$$

The overall probability of error will decrease after every iteration and the overall BER will reduce

LZW-

This method was developed originally by Ziv and Lempel, and subsequently improved by Welch. LZW compression is the compression of data or a file into smaller size using the concept of a lookup table. A particular LZW compression algorithm takes each input sequence of bits of a given length and creates an entry in a table (or dictionary) for that particular bit pattern, consisting of the pattern itself and a shorter code. As input is read, any pattern that has been read before results in the substitution of the shorter code, effectively compressing the total amount of input to something smaller.

HUFFMAN ENCODING

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

There are mainly two major parts in Huffman Coding:
1) Build a Huffman Tree from input characters.
2) Traverse the Huffman Tree and assign codes to characters.

Steps to be followed for Huffman encoding:

- Count the frequency of each character in the file to be encoded.
- Extract two nodes with the minimum frequency from the min heap.
- Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child.
- Repeat the above steps till the coding is finished for all the given input sequence.

SHIFT ENCRYPTION

For the Shift encryption. The overall mechanism uses a binary shifter block both at the transmitter and receiver. The overall system plays along the binary bit blocks that need to be transmitted. Suppose a code word has length 'k' and 'n' blocks are to be transmitted. The shift encoder will use the 'k-3' bits of the (n-1)th block to find the decimal equivalent of the bits associated with that block.

Suppose the (n-1)th block has data like- 1001. The last (k-3)bits give the data as 011 which has decimal equivalent to 1. Now the nth block suppose has data 1100. This block of code will undergo shifting according to the decimal data given by the prior block. In this case it is 1. Thus the new block will be rotated left by 1 bit to give 1001. Now this new encrypted bit block will be sent and at the receiver the reverse operation will rotate right the incoming block by 1 bit due to the data incurred by the previous block. Thus the data can again be decoded as 1100.

CODES-

Code 1-

```
clear all;
clc; close
all;
global n k L Type no_of_symbols_in M N S SymbolMapping number_of_levels
bits_per_symbol; global Codeword Nextstate Previousstate
Interleave_mode_in

n=input('Enter the length of dataword: ')%n=2; k=input('Enter
the length of the codeword: ')%k=3;
L=input('Enter the Memory stages: ')%L=3;
Type='PSK'; no_of_symbols_in=512;
Interleave_mode_in='OFF';

%Generation of Trellis To begin the simulation
[Codeword, Nextstate, Previousstate]=genpoly(n,k,L);



M=bitshift(1,n);
N=no_of_symbols_in;
S=bitshift(1,L);

[SymbolMapping,bits_per_symbol,number_of_levels] = Modulation_SP(Type,k);
```

```
Ec=1; snr=0; snr_stop=4;
no_of_info_bits=no_of_symbols_in*n;
no_of_coded_bits=no_of_symbols_in*k;
b_Input_bits=input('Enter the bit-stream to be encoded: ');
figure;     axes('YScale','log');     title('TCM BER');
xlabel('Eb/No in dB');     ylabel('BER');     count=1;
while (snr <= snr_stop)



    % Counters to store the number of errors and bits simulated so far.

    error_counts=0;
bit_count=0;
SNR=snr-10*log10(n);
snrnotdb=10^(snr/10);
c=0;

    while (error_counts < 1000 && c <=10000)

        %Compute energy per bit
        Eb=Ec/((no_of_info_bits/no_of_coded_bits)*bits_per_symbol);
        %variance
        N0=Eb*(snrnotdb^-1);
        b_Input_bits=round(rand(1,no_of_info_bits));
        symbols =bits_to_symbol(n,no_of_symbols_in,b_Input_bits);

[Output_symbols,Tx_signal]=encode_symbols(symbols,Codeword,Nextstate,Symbol
Mapping);
        Rx_signal=Tx_signal+
sqrt(N0/2)*(randn(size(Tx_signal))+1i*randn(size(Tx_signal)));
        Pr=demodulate_soft_symbols(Rx_signal,N0);
b_decoded_bits=decode_bits(Pr);
        total_error=sum(b_decoded_bits ~= b_Input_bits);
                    error_counts=error_counts+total_error;
bit_count=bit_count+sum(b_decoded_bits                ~=
b_Input_bits)+sum(b_decoded_bits    ==    b_Input_bits);
c=c+1;

end

ber=error_counts/bit_count;

    results(count,1)=snr;
results(count,2)=ber;

    %plot the BER
semilogy(results(:,1),results(:,2));
%print the values to txt file     fprintf('!%f
\t\t %f \n',snr,ber);

snr=snr+0.5;
count=count+1;
end
```

CODE 2-

```
%LZW encoding

%convert LZW to 5 bits clc; clear all; close all; x=input('Enter the
encoding format: 1->LZ78 | 2->Huffman') if(x==1) datain=input('enter
the string in single quote with symbol $ as End of string =');%input
data
```

```
lda=length(datain);%length of datainput dictionary=input('enter the
dictionary in single quote(symbol used in string are to be
included)=');%input dictionary

ldi=length(dictionary);%length of dictionary
j=1;%used for generating code n=0;%used for
%loop used for string array to cell array conversion for
i=1:ldi
dictnew(i)={dictionary(i)};
end
p=datain(1);%first symbol s=p;%current symbol k=1;
%used for generating transmitting output code
i=1;%for loop m=0;
while datain(i)~= '$'%end of symbol
c=datain(i+1); if c~='$'
comb=strcat(s,c);%just for see combination if
strcmp(dictnew,strcat(s,c))==0
dictnew(j+ldi)={strcat(s,c)};
%lopp and check used for generating transmitting
%code array
check=ismember(dictnew,s);
for l=1:length(check) if
check(l)==1 tx_trans(k)=l;
k=k+1; break; end end
s=c; j=j+1; i=i+1; m=m+1;
else s=strcat(s,c);
i=i+1; end else
%for sending last and eof tx_trans
check=ismember(dictnew,s); for
l=1:length(check) if check(l)==1
tx_trans(k)=l; k=k+1;
tx_trans(k)=0; end end break; end
end
display('new dictionary=')
display(dictnew); display(tx_trans);
%each block will get the bits rotated by the last 3 bits of the previous
%block
%--------------------------------------------------------------------------
new_arr=de2bi(tx_trans)
[r,c]=size(new_arr) %encode[r:c]
for i=1:1:r     if(i==1)
encode(1,:)=new_arr(1,:);
else
        if(new_arr((i-1),2)==0 && new_arr((i-1),3)==0&& new_arr((i-
1),4)==0)
        encode(i,:)=circshift(new_arr(i,:),[0,0]);
end
        if(new_arr((i-1),2)==0 && new_arr((i-1),3)==0&& new_arr((i-
1),4)==1)
        encode(i,:)=circshift(new_arr(i,:),[0,1]);
end
        if(new_arr((i-1),2)==0 && new_arr((i-1),3)==1&& new_arr((i-
1),4)==0)
        encode(i,:)=circshift(new_arr(i,:),[0,2]);          end
if(new_arr((i-1),2)==0 && new_arr((i-1),3)==1&& new_arr((i-
1),4)==1)
        encode(i,:)=circshift(new_arr(i,:),[0,3]);          end
if(new_arr((i-1),2)==1 && new_arr((i-1),3)==0&& new_arr((i-
1),4)==0)
        encode(i,:)=circshift(new_arr(i,:),[0,4]);          end
if(new_arr((i-1),2)==1 && new_arr((i-1),3)==0&& new_arr((i-
1),4)==1)
        encode(i,:)=circshift(new_arr(i,:),[0,5]);          end
if(new_arr((i-1),2)==1 && new_arr((i-1),3)==1&& new_arr((i-
1),4)==0)
```

```
        encode(i,:)=circshift(new_arr(i,:),[0,6]);
end
        if(new_arr((i-1),2)==1 && new_arr((i-1),3)==1&& new_arr((i-
1),4)==1)
        encode(i,:)=circshift(new_arr(i,:),[0,7]);
end     end             encode

end
%decode  decode=bi2de(encode)

%LZ79 for the crypted message
  dicgen=dictionary;
ldgen=length(dicgen);
ldtx=length(tx_trans);
index=length(dictionary);
string='';
%loop and below inst. used for cell array to char array
dicgen=cellstr(dictionary);  for i=1:ldi
dicgen(i)={dictionary(i)};  end  g=1;
entry=char(dictionary(tx_trans(1)));%first symbol  g=g+1;%
next symbol
while decode(g)~=0 %for EOF  s=entry;
           entry=char(dicgen(decode(g)));
string=strcat(string,s);  %detected string
index=index+1;  % next index
dicgen(index) = {strcat(s,entry(1))};%upgrade dictionary  g=g+1;
% next index
  end
string=strcat(string,entry)  disp(dicgen);
disp('The original input without encryption was: ');
disp(datain)
display('received original string=');
disp(string);
fid=fopen('data.txt','w');
fprintf(fid,'%s',string);
fclose(fid);  end    if(x==2)
  s=input('input string : ','s')
orig=s;    for i=1:length(s)
x(i)=abs(96-double(s(i)));
    %x(i)=tt;
end
  y=zeros(1,26);  for
i=1:length(x)
y(x(i))=y(x(i))+1;  end
z=zeros(1,26);  for
i=1:length(z)
if(y(i)>0)
z(i)=1;      end end z
k=1;  for i=1:26
if(z(i)==1)
new(k)=i+96;
k=k+1;      end end    new
str=char(new)
l1=length(s)  k=1;  for
i=1:26      if  (y(i)>0)
prob(k)=y(i)./l1;
k=k+1;      end end
prob      my_str=str

prob_dist=prob
num_bits = ceil(log2(length(prob_dist)))

disp('Character  Probability:');  for  i  =  1:length(prob_dist)
display(strcat(my_str(i),' -->  ',num2str(prob_dist(i))));  end
total = sum(prob_dist)
```

```
for   i   =   1:length(my_str)
sorted_str{i} = my_str(i); end
init_str = sorted_str init_prob
= prob_dist

sorted_prob = prob_dist  rear =
1; while (length(sorted_prob) >
1)
    [sorted_prob,indeces] = sort(sorted_prob,'ascend');
sorted_str  =  sorted_str(indeces);          new_node  =
strcat(sorted_str(2),sorted_str(1));         new_prob  =
sum(sorted_prob(1:2));                    sorted_str    =
sorted_str(3:length(sorted_str));         sorted_prob  =
sorted_prob(3:length(sorted_prob));        sorted_str  =
[sorted_str, new_node];     sorted_prob = [sorted_prob,
new_prob];                newq_str(rear)   =   new_node;
newq_prob(rear) = new_prob;      rear = rear + 1; end
tree = [newq_str,init_str] tree_prob
= [newq_prob, init_prob]
[sorted_tree_prob,indeces] = sort(tree_prob,'descend'); sorted_tree
= tree(indeces);
  parent(1) = 0; num_children
=     2;      for     i     =
2:length(sorted_tree)

    me = sorted_tree{i};     count = 1;
parent_maybe = sorted_tree{i-count};
diff = strfind(parent_maybe,me);
while (isempty(diff))          count =
count + 1;
        parent_maybe = sorted_tree{i-count};
        diff = strfind(parent_maybe,me);     end
parent(i) = i - count; end   treeplot(parent);
title(strcat('Huffman Coding Tree - "',my_str,'"'));
  display(sorted_tree)
display(sorted_tree_prob)

[xs,ys,h,s] = treelayout(parent); text(xs,ys,sorted_tree);

for i = 2:length(sorted_tree)     my_x = xs(i);
my_y = ys(i);      parent_x = xs(parent(i));
parent_y = ys(parent(i));      mid_x = (my_x +
parent_x)/2;      mid_y = (my_y + parent_y)/2;
slope  = (parent_y - my_y)/(parent_x - my_x);
if (slope > 0)          weight(i) = 0;      else
weight(i) = 1;      end
    text(mid_x,mid_y,num2str(weight(i)));
end
for i = 1:length(sorted_tree)
code{i} = '';      index = i;      p
= parent(index);      while(p ~= 0)
w = num2str(weight(index));

        code{i} = strcat(w,code{i});
index = parent(index);          p =
parent(index);      end end    k=1;
for i= 1:length(sorted_tree)
t=double(char(sorted_tree(i)));
if (length(t)==1)
result(k)=sorted_tree(i);
fin_index(i)=i;           k=k+1;
end end result    k=1;
for i=1:length(sorted_tree)
if(fin_index(i)>0)
```

```
f_index(k)=fin_index(i);
k=k+1;      end
end
for i=1:length(f_index)
final_code_word(i)=code(f_index(i));
end disp('final')  final_code_word

huff_arr=char(cellstr(final_code_word))
%huff_arr_new=char2bi(huff_arr)
[r,c]=size(huff_arr)

%-----------------------------------
for i=1:1:r      if(i==1)
        encode_huff(1,:)=huff_arr(1,:);      else
if(huff_arr((i-1),2)==0 && huff_arr((i-1),3)==0&& huff_arr((i-
1),4)==0)
        encode_huff(i,:)=circshift(huff_arr(i,:),[0,0]);
end
        if(huff_arr((i-1),2)==0 && huff_arr((i-1),3)==0&& huff_arr((i-
1),4)==1)
        encode_huff(i,:)=circshift(huff_arr(i,:),[0,1]);
end
        if(huff_arr((i-1),2)==0 && huff_arr((i-1),3)==1&& huff_arr((i-
1),4)==0)
        encode_huff(i,:)=circshift(huff_arr(i,:),[0,2]);
end
        if(huff_arr((i-1),2)==0 && huff_arr((i-1),3)==1&& huff_arr((i-
1),4)==1)
        encode_huff(i,:)=circshift(huff_arr(i,:),[0,3]);
end
        if(huff_arr((i-1),2)==1 && huff_arr((i-1),3)==0&& huff_arr((i-
1),4)==0)
        encode_huff(i,:)=circshift(huff_arr(i,:),[0,4]);          end
if(huff_arr((i-1),2)==1 && huff_arr((i-1),3)==0&& huff_arr((i-
1),4)==1)
        encode_huff(i,:)=circshift(huff_arr(i,:),[0,5]);          end
if(huff_arr((i-1),2)==1 && huff_arr((i-1),3)==1&& huff_arr((i-
1),4)==0)
        encode_arr(i,:)=circshift(huff_arr(i,:),[0,6]);          end
if(huff_arr((i-1),2)==1 && huff_arr((i-1),3)==1&& huff_arr((i-
1),4)==1)
        encode_huff(i,:)=circshift(huff_arr(i,:),[0,7]);
end     end             encode_huff

end
fid=fopen('data.txt','w');
fprintf(fid,'%s',huff_arr);  fclose(fid);
%---------------------------------
%huffman_code=[result',final_code_word']
```

OUTPUT

```
Enter the length of dataword: 2
n =

     2

Enter the length of the codeword: 3
k =

     3

Enter the Memory stages: 3
L =

     3
```

```
Enter the bit-stream to be encoded: [1 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1]
0.000000        0.189128
0.500000        0.168945
1.000000        0.136108
1.500000        0.122803
2.000000        0.083415
2.500000        0.070661
3.000000        0.039766
3.500000        0.021675
4.000000        0.010244
4.500000        0.003634
5.000000        0.001217
```

EXPECTED

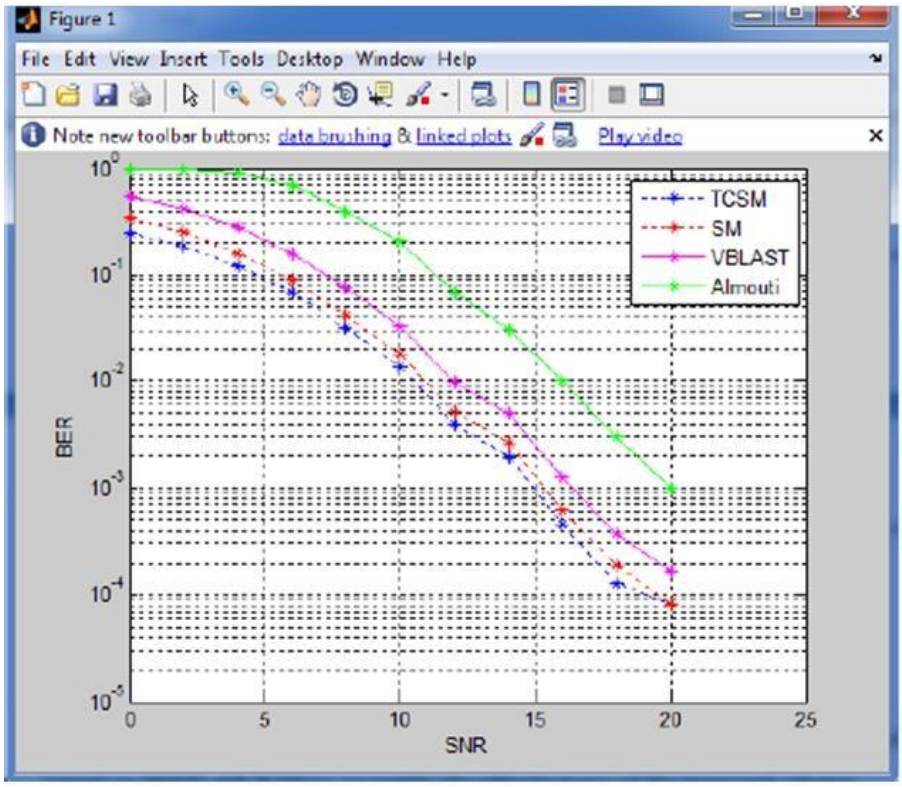


Figure 1
File Edit View Insert Tools Desktop Window Help
Note new toolbar buttons: data brushing & linked plots
Play video
TCSM
SM
VBLAST
Almouti
BER
SNR
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
0
5
10
15
20
25

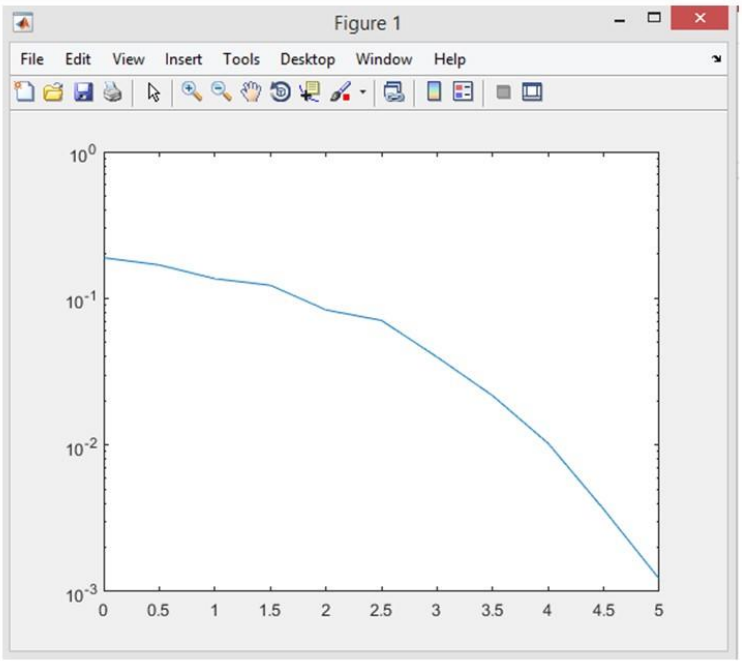


Figure 1
File Edit View Insert Tools Desktop Window Help
10^0
10^-1
10^-2
10^-3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Results-

```
Enter the encoding format: 1->LZ78 | 2->Huffman1

x =

     1

enter the string in single quote with symbol $ as End of string ='abbcdbabdbbabbaccbd$'
enter the dictionary in single quote(symbol used in string are to be included)='abcd'
new dictionary=

dictnew =

  Columns 1 through 15

    'a'    'b'    'c'    'd'    'ab'    'bb'    'bc'    'cd'    'db'    'ba'    'abd'    'dbb'    'bab'    'bba'

  Columns 16 through 18

    'cc'    'cb'    'bd'
```

```
tx_trans =

     1     2     2     3     4     2     5     9    10     6     1     3     3     2     4     0


new_arr =

     1     0     0     0
     0     1     0     0
     0     1     0     0
     1     1     0     0
     0     0     1     0
     0     1     0     0
     1     0     1     0
     1     0     0     1
     0     1     0     1
     0     1     1     0
     1     0     0     0
     1     1     0     0
     1     1     0     0
     0     1     0     0
```

```
encode =

     1     0     0     0
     0     1     0     0
     0     1     0     0
     1     1     0     0
     0     0     1     0
     0     0     0     1
     1     0     1     0
     0     1     1     0
     1     0     1     0
     0     0     1     1
     0     0     1     0
     1     1     0     0
     1     1     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     0
```

```
decode =

     1
     2
     2
     3
     4
     8
     5
     6
     5
    12
     4
     3
     3
     2
     4
     0
```

```
string =

abbcdcdabbbabbbadccbd

  Columns 1 through 14

    'a'    'b'    'c'    'd'    'ab'    'bb'    'bc'    'cd'    'dc'    'cda'    'abb'    'bba'    'abb'    'bbad'

  Columns 15 through 18

    'dc'    'cc'    'cb'    'bd'

The original input without encryption was:
abbcdbabdbbabbaccbd$
received original string=
abbcdcdabbbabbbadccbd
```

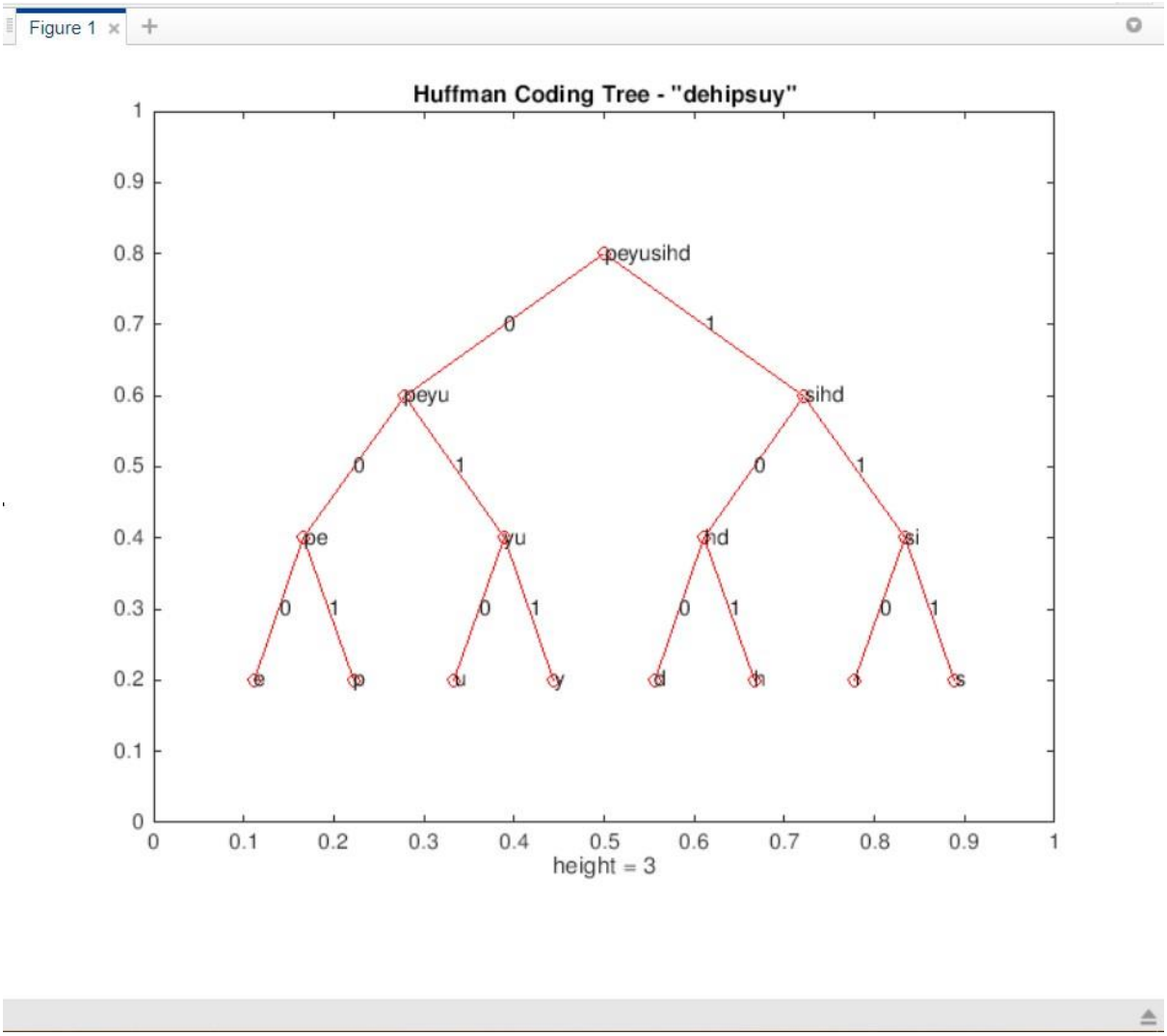
```
Enter the encoding format: 1->LZ78 | 2->Huffman
2

x =

     2

input string :
piyushdeep


s =

    'piyushdeep'



z =

     0     0     0     1     1     0     0     1     1     0     0     0     0
0     0     1     0     0     1     0     1     0     0     0     1     0
```

```
new =

   100   101   104   105   112   115   117   121


str =

    'dehipsuy'


l1 =

    10


prob =

    0.1000    0.2000    0.1000    0.1000    0.2000    0.1000    0.1000
0.1000


my_str =

    'dehipsuy'


prob_dist =

    0.1000    0.2000    0.1000    0.1000    0.2000    0.1000    0.1000
0.1000



num_bits =

     3

Character Probability:
d -->0.1
e -->0.2
h -->0.1
i -->0.1
p -->0.2
s -->0.1
u -->0.1
y -->0.1

total =

    1.0000
```

```
init_str =

  1×8 cell array

    {'d'}    {'e'}    {'h'}    {'i'}    {'p'}    {'s'}    {'u'}    {'y'}


init_prob =

    0.1000    0.2000    0.1000    0.1000    0.2000    0.1000    0.1000
0.1000


sorted_prob =

    0.1000    0.2000    0.1000    0.1000    0.2000    0.1000    0.1000    0.1000


tree =

  1×15 cell array

    {'hd'}    {'si'}    {'yu'}    {'pe'}    {'sihd'}    {'peyu'}
{'peyusihd'}    {'d'}    {'e'}    {'h'}    {'i'}    {'p'}    {'s'}    {'u'}
{'y'}


tree_prob =

    0.2000    0.2000    0.2000    0.4000    0.4000    0.6000    1.0000
0.1000    0.2000    0.1000    0.1000    0.2000    0.1000    0.1000    0.1000


sorted_tree =

  1×15 cell array

    {'peyusihd'}    {'peyu'}    {'pe'}    {'sihd'}    {'hd'}    {'si'}
{'yu'}    {'e'}    {'p'}    {'d'}    {'h'}    {'i'}    {'s'}    {'u'}    {'y'}


sorted_tree_prob =

    1.0000    0.6000    0.4000    0.4000    0.2000    0.2000    0.2000
0.2000    0.2000    0.1000    0.1000    0.1000    0.1000    0.1000    0.1000
```

```
result =

  1×8 cell array
    {'e'}    {'p'}    {'d'}    {'h'}    {'i'}    {'s'}    {'u'}    {'y'}


final

final_code_word =

  1×8 cell array

    {'000'}    {'001'}    {'100'}    {'101'}    {'110'}    {'111'}    {'010'}
{'011'}


huff_arr =

  8×3 char array


    '000'
    '001'
    '100'
    '101'
    '110'
    '111'
    '010'
    '011'


r =

     8


c =

     3


encode_huff =

    '000'


encode_huff =

    '000'
```

```
encode_huff =

    '000'


encode_huff =

    '000'


encode_huff =

    '000'


encode_huff =

    '000' encode_huff =

    '000'


encode_huff =

    '000'
```



CONCLUSION

Here, a new bandwidth efficient coding technique for spatial modulation is implemented. This scheme consists of trellis coded modulation applied to the antenna index of the spatial modulation. Trellis coded modulation is a method which improves the noise immunity of digital transmission systems without expansion of bandwidth or reduction of data rate. In this project models of the communication channel were presented, using a combination of convolution coding and one of the selected modulation schemes. Also the BER curve of TCSM for different modulation scheme is obtained.

In Shift Encoder Encryption the decimal equivalent value is used to shift the nth block by exactly the number. This process continues till all the bits are transmitted. This procedure is safe as even if a block of code is captured by someone illegally. That person cannot decrypt the message without knowing which block of data was transmitted before it and which was transmitted even before it. So the decryption needs all the blocks at once.

REFERENCES

- https://descanso.jpl.nasa.gov/monograph/series3/complete1.pdf
- https://www.researchgate.net/publication/301911935_A_bandwidth_efficient_coding_technique_for_spatial_modulation_and_its_error_performance
- http://inpressco.com/wp-content/uploads/2016/04/Paper36590-595.pdf
- http://www.complextoreal.com/wp-content/uploads/2013/01/tcm.pdf
- A bandwidth Efficient Coding Technique for Spatial modulation and its error performance by Basil. K. Jeemon (ECE Dept.) FISAT, University of Kerala, India
- Principles of TCM by Charan Langton 2004
- RC encryption By Ronald L. Rivest. MIT Laboratory of Computer Science, 545 Technology Square, Cambridge.