Multimedia and Video Communications (SSY150)
Lab 3

Group 5: Haitham Babbili, Olalekan Peter Adare

18th May 2020

Task 1

1

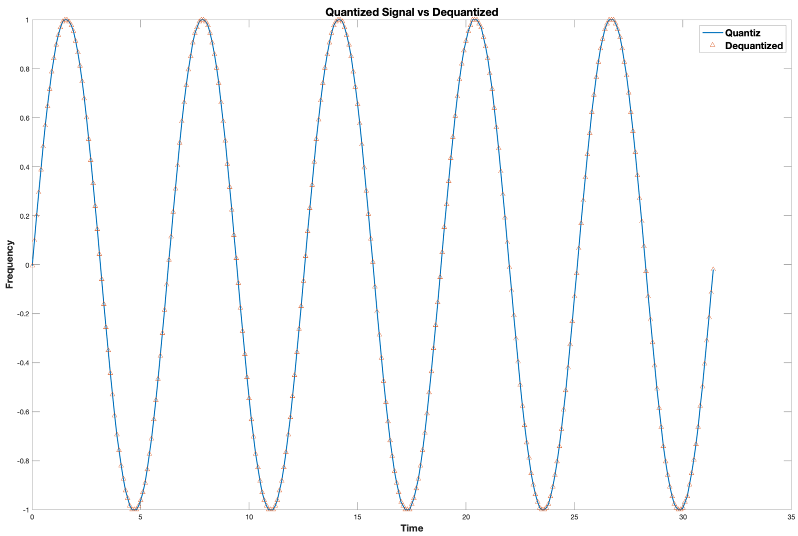

Figure 1:

Task 2

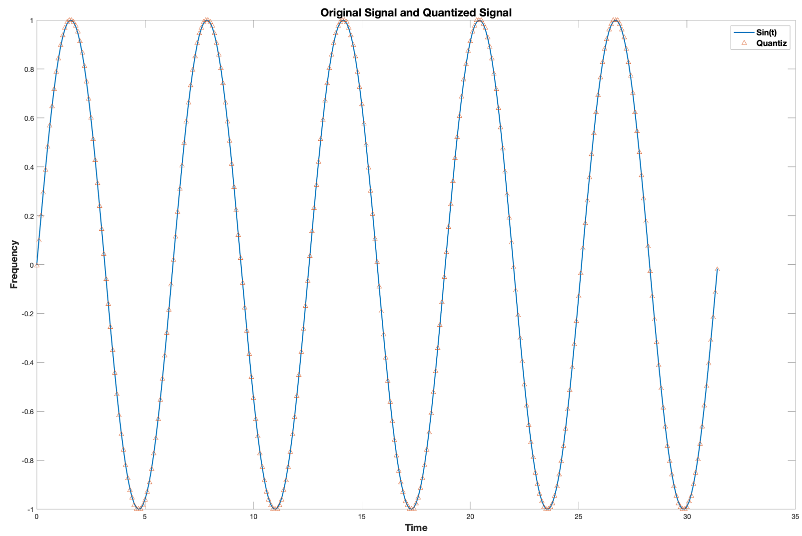

Figure 2:

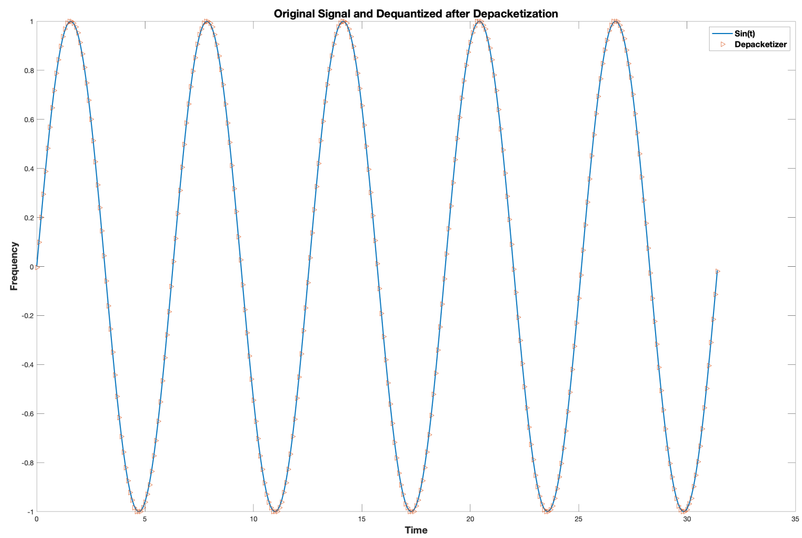

Figure 3:

Comparing figures 2 and 3 above, the plots are the same. This implies that the packetization and depacketization process is correctly implemented.

Task 3

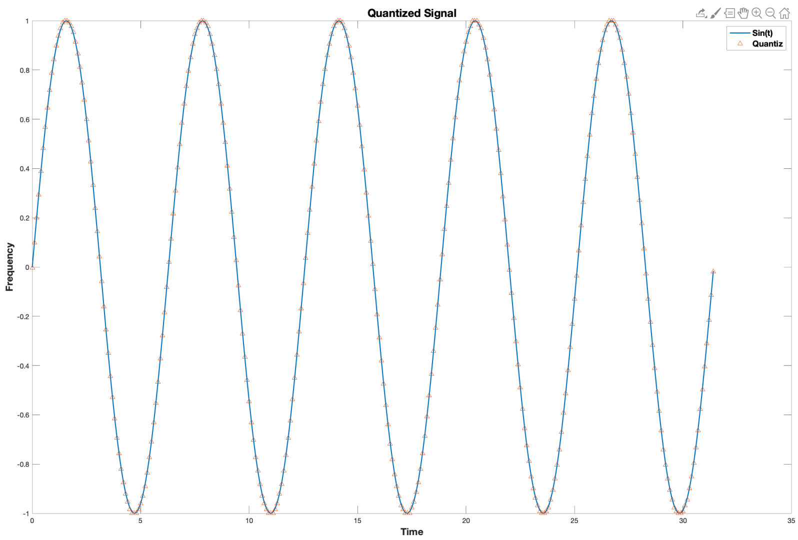

Figure 4:

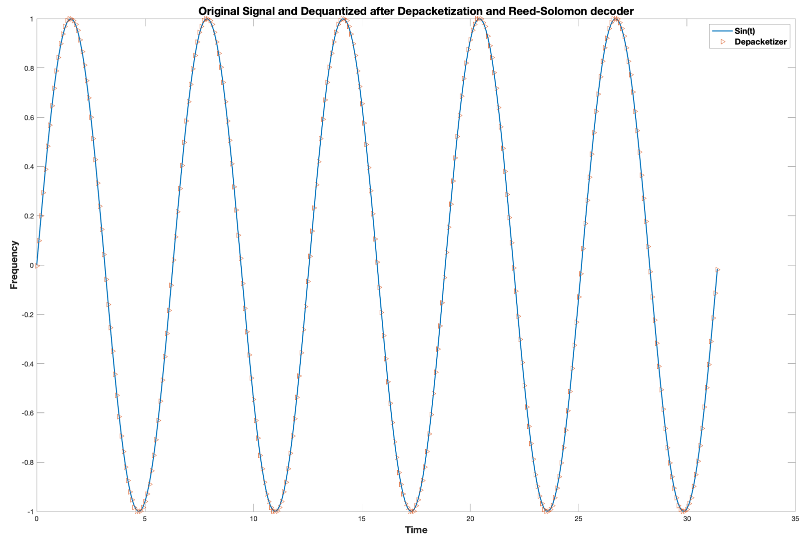

Figure 5:

Comparing figures 4 and 5 above, the plots are the same. This implies that the quantization, packetization and reed-solomon coding can be correctly decoded, depacketized and de-quantized at the receiver.

Task 8

REPORT

a.1

Figure 6:

Here, this test was performed without the interleaver/de-interleaver and with a given packet loss of 3% .It is observed that the reconstructed image had like 8 missing blocks. The reed-solomon code given is RS(255,127). This means it can correct $[\frac{n-k}{2}] = [\frac{255-127}{2}]$= 64 symbol errors can be maximally corrected. 259 packets were transmitted, with 8 packets lost during transmission, based on the given 3% packet loss information.This means there will be 8 x 127 = 1016 symbols in error. This gives corrupted packets in the transmission.The RS code can only correct for 64 error-ed symbols. The related codewords in the packets that have been lost, or corrupted, cannot be recovered. This is why the reconstructed image has some missing blocks.

a.2

Figure 7:

Here, this test was performed with the interleaver/de-interleaver and with a given packet loss of 3% . It was observed that the reconstructed image was far better with no visible missing blocks, quite close to the original image, but with a lower quality compared to the original image since it was a lossy compression, based on DCT and the data compression ratio. It also means it was easier for the RS coder to correct errors across all the codewords. There was no corrupted packet, hence a very good reconstructed image was generated.

a.3

The interleaver spreads the errors uniformly across the codewords. The matrix interleaver improves the performance of the RS code by enabling efficient correction of errors across all the symbols and packets, and not just on a few. There were no corrupted packets and this is why the reconstructed image had no missing blocks. The downside of the use of the matrix interleaver is extra delay and memory buffer requirement, needed to temporally store several packets. Therefore, without using the matrix interleaver, there will be corrupted packets and the errors will not be uniformly spread out. This will lead to visible missing blocks in the reconstructed image.

b.1

Figure 8:

b.2

Figure 9:

b.3

In b1, the reconstructed image was with good quality with no visible block lost in the image. This was without the interleaver/de-interleaver. This is because the codewords have not be redistributed and all the errors are equally spread across the rows. Since t=60 and RS can correct a maximum of 64 symbol errors, thus all errors were corrected and cleared out in the reconstructed image.
Whereas in b2, with the interleaver and de-interleaver being introduced, the reconstructed image had a lot of visible corrupted blocks. There is random noise generated and spread in the channel. The interleaver mixes up symbols in the codewords. At the receiver side, the de-interleaver re-arranged the codewords, where some codewords had less than 64 and the RS code corrected them, while some had more than 64 symbol errors, which the RS code could not correct. Therefore, the reconstructed image had some pixel blocks correctly displayed while the rest are corrupted blocks.

b.4

Figure 10:

With t=100 and no interleaver/de-inteleaver, the RS code can correct a maximum of 64 symbol errors. Here, there are now 100 symbol errors per row, which is above the error correcting capability of the RS code, hence the reconstructed image is totally lost as compared to the original image.
PSNR= -9.8100752261o47600
MSSIM= 3.875050975342177e-04

c.1

For packet losses:

For (a.1)
PSNR= 6.162350608305883 dB
MSSIM= 0.790218517489402

For (a.2)
PSNR=31.424888896431150 dB
MSSIM= 0.876557114102985

Comparing the PSNR and MSSIM values,these values for (a.2) are higher than the values from (a.1). Specifically, the PSNR value from (a.2) is above 30 which indicates a very good quality image. With the use of the interleaver, it gave a far better reconstructed image quality.

c.1

For symbol losses: For (b.1)
PSNR= 31.424888896431150 dB
MSSIM= 0.876557114102985

For (b.2)
PSNR= -2.256475115776966 dB
MSSIM= 0.372465798638101

Comparing the PSNR and MSSIM values,these values for (b.1) are higher than the values from (b.2). Specifically, the PSNR value from (b.1) is above 30 which indicates a very good quality image. With the introduction of the interleaver, the quality of the reconstructed image did not improve; it was worse. The interleaver ended up just spreading more errors, symbol/bit errors, above the error correcting capability of the RS code.

Report

1

In the packet loss case, the matrix interleaver at the receiver gave a very good quality reconstructed image. This is because the errors were uniformly spread across all packets, and RS code corrected all the errors. Whereas, for the symbol error case, the matrix interleaver gave a worse reconstructed image because the error spread were above the RS code error limit in different block pixels.

2

In the source compression, a global threshold was used for the DCT.
The main difference is that the global threshold, a threshold value is defined and is applied to the all blocks in the image. Global threshold is as good as the degree of intensity separation between the two peaks in the image. It is an unsophisticated segmentation approach. Whereas, the local threshold chooses different threshold values for every pixel in the image based on an analysis of its neighboring pixels. The local threshold gives more details in the reconstructed image.
The complication with global threshold is first the determination of the threshold value to be used,

so as not to lose useful information, depicted as just black or white, due to illumination.The image may have multiple threshold peaks. With global threshold approach, some details are lost in the reconstructed image.

3

3.1

With a wired network, a new block will be introduced called transport protocols. This will represent all devices that implement specific transport communication protocols in the wired IP network, like IP routing protocols (BGP,RIP,OSPF,ISIS), TCP, UDP, RSVP and others. This includes routers, switches, firewall, to mention a few. At the physical layer, there can be modems and multiplexers/de-multiplexers as well.

3.2

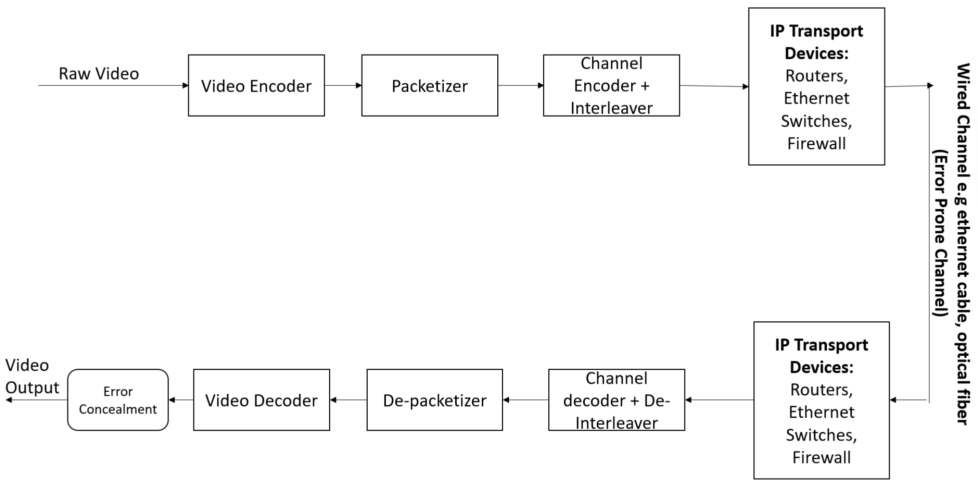

Figure 11:

3.3

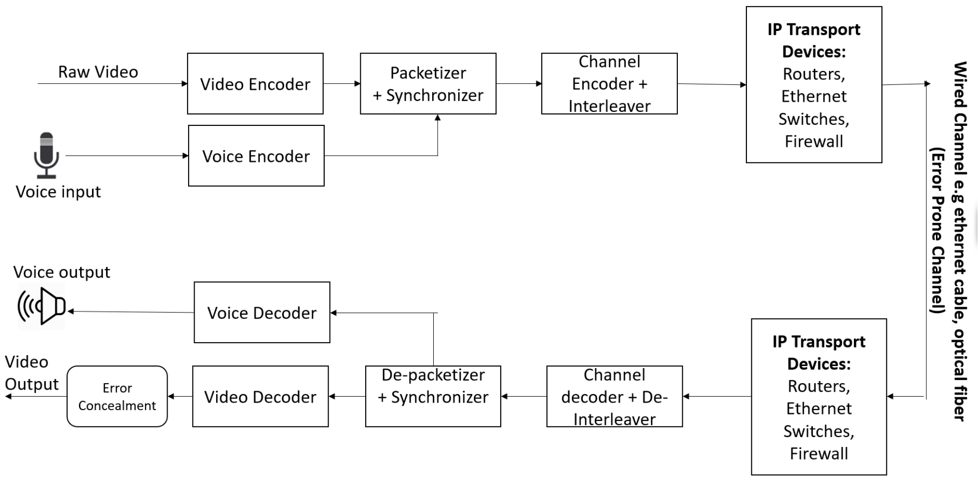
Raw Video
Video Encoder
Packetizer + Synchronizer
Channel Encoder + Interleaver
IP Transport Devices: Routers, Ethernet Switches, Firewall
Voice Encoder
Voice input
Wired Channel e.g ethernet cable, optical fiber (Error Prone Channel)
Voice output
Voice Decoder
Video Output
Error Concealment
Video Decoder
De-packetizer + Synchronizer
Channel decoder + De-Interleaver
IP Transport Devices: Routers, Ethernet Switches, Firewall

Figure 12:

Matlab Code

```
%% Task 1
close all;
clear all;
clc

t=[0:0.1:10*pi];
signal=sin(t);

[index,quantized,codebook]=quantization(signal);
recon_signal=codebook(index+1);
figure()
set(gcf, 'Position', [100, 100, 1420, 960])
plot(t,quantized,'linewidth',1.5)
hold on
plot(t,recon_signal,'^')
xlabel('Time','FontSize',14,'FontWeight','bold')
ylabel('Frequency','FontSize',14,'FontWeight','bold')% step 1.4
title('Quantized Signal vs Dequantized','FontSize',16,'FontWeight','bold')
legend('Quantiz','Dequantized','FontSize',14,'FontWeight','bold')
```

```
%% Task 1
close all;
clear all;
clc

t=[0:0.1:10*pi];
signal=sin(t);

[index,quantized,codebook]=quantization(signal);


figure()
set(gcf, 'Position', [100, 100, 1420, 960])
plot(t,signal,'linewidth',1.5)
hold on
plot(t,quantized,'^')
xlabel('Time','FontSize',14,'FontWeight','bold')
ylabel('Frequency','FontSize',14,'FontWeight','bold')
title('Original Signal and Quantized Signal','FontSize',16,'FontWeight','bold')
legend('Sin(t)','Quantiz','FontSize',12,'FontWeight','bold')
%% Task 2
k=127;
packets = packetizer(index,k);

% Depacketizer
```

```matlab

depackets = depacketizer(packets,index);

% Dequantized
recon_signal = codebook(depackets+1);

figure()
set(gcf, 'Position', [100, 100, 1420, 960])
plot(t,signal,'linewidth',1.5)
hold on
plot(t,recon_signal,'>')
xlabel('Time','FontSize',14,'FontWeight','bold')
ylabel('Frequency','FontSize',14,'FontWeight','bold')
title('Original Signal and Dequantized after Depacketization','FontSize',16,'FontWeight','bold')
legend('Sin(t)','Depacketizer','FontSize',12,'FontWeight','bold')
```

```matlab
%% Task 1
close all;
clear all;
clc

t=[0:0.1:10*pi];
signal=sin(t);

[index,quantized,codebook]=quantization(signal);


figure()
set(gcf, 'Position', [100, 100, 1420, 960])
plot(t,signal,'linewidth',1.5)
hold on
plot(t,quantized,'^')
xlabel('Time','FontSize',14,'FontWeight','bold')
ylabel('Frequency','FontSize',14,'FontWeight','bold')
title('Quantized Signal','FontSize',16,'FontWeight','bold')
legend('Sin(t)','Quantiz','FontSize',12,'FontWeight','bold')
%% Task 2
k=127;                                   % Message length
packets = packetizer(index,k);

%% Task 3
% Reed Solomon encoder
m=8;                                     % Bits per symbol
n=2^m 1;                                 % Codeword length
[codes msgwords] = rs_code(packets,m,k); % RS encoding

%% Reed Solomon decoder
```

```
dec_msg=rsdec(codes,n,k);                    % decoded packets
isequal(dec_msg,msgwords);

%% depacketizer

depackets = depacketizer(dec_msg,index);

%% Dequantized
recon_signal = codebook(depackets.x+1);

figure()
set(gcf, 'Position', [100, 100, 1420, 960])
plot(t,signal,'linewidth',1.5)
hold on
plot(t,recon_signal,'>')
xlabel('Time','FontSize',14,'FontWeight','bold')
ylabel('Frequency','FontSize',14,'FontWeight','bold')
title('Original Signal and Dequantized after Depacketization and Reed
    Solomon decoder','FontSize',16,'FontWeight','bold')
legend('Sin(t)','Depacketizer','FontSize',12,'FontWeight','bold')
```

```
close all;
clear all;
clc

imag= imread('lena.bmp');    % Read the image
Imag=mat2gray(imag);         % Converting image format to intensity image
figure()
imshow(Imag)                 % Displaying image
title('Original Image')

%% DCT Block1
[x,y]=size(Imag);
block_size=16;
comp_ratio=0.5;                          % compression ratio
pl_rate=0.03;                            % Packet loss rate
N=16*16;                                 % pixels in one block
N1=round(comp_ratio*16^2);               % discarded dct coeficiences
Nc=N N1;

% apply dct
dct = zeros(size(Imag));
for i=1:x/16
    for j=1:y/16
        block=Imag((i 1)*16+1:i*16,(j 1)*16+1:j*16);
        Block=dct2(block);
        dct((i 1)*16+1:i*16,(j 1)*16+1:j*16)=Block;
    end
end


% zigzag scanning

signal=[];
for i=1:x/16
    for j=1:y/16
        blocks=dct((i 1)*16+1:i*16,(j 1)*16+1:j*16);
        blocks_skem=zigzag(blocks);                      % Skem all blocks
        sequens=blocks_skem(1:Nc);
        signal=[signal,sequens];
    end
end

%% Quantization block

[index,quantized,codebook]=quantization(signal);


%% Packet
```

```

k=127;
packets = packetizer(index,k);

%% Reed Solomon encoder

m=8;  %Bits per symbol
n=2^m 1;
[codes msgwords codewords] = rs_code(packets,m,k);

[nw, nl]=size(codes);


%% Interleaving

% intrlv = interlever(codes);

%% Channel
method=input('Please enter 1 for packet loss or 2 for bit error:');
switch(method)


    case{1}
        %5.1
%          codes_noisy=rsenc(msgwords, n, k);
%          codes_noisy=intrlv; % with interlever
        codes_noisy=codes; % without interlever
        e_packet=zeros(1, n);
        errorpacket=gf(e_packet,m);
        n_erropack=round(nw*pl_rate);
        loss_packets=randi([1,nw],1,n_erropack);
        for i=loss_packets
            codes_noisy(i, :) = errorpacket;
        end
%          [dc,nerrs,corrcodes]=rsdec(codes_noisy, n, k);

        case{2}
        % step5.2
%          codes = rsenc(msgwords, n, k);
%          codes= intrlv; % with interlever
        error_detect=floor((n k)/2);
%          t=randi([0,error_detect],1);
%          t=60;
        t=100;
        noise =randi( 2^m  1,nw,n).* randerr(nw, n, t);
        codes_noisy = codes + noise;
        [dc,nerrs,corrcodes]=rsdec(codes_noisy, n, k);
end
```

```
%% Deinterleaving

% deintr = deinterleaver(codes_noisy,nw,nl);

%% Reed Solomon decoder
% dec_msg=rsdec(deintr,n,k);% decoded packets with interleaver

dec_msg=rsdec(codes_noisy,n,k);% decoded packets without interleaver

isequal(dec_msg,codewords);


%% Depacket

depackets = depacketizer(dec_msg,index);

%% Dequantizer

% quantized=codebook(depackets+1);

quantized=codebook(depackets.x+1);

%%

% apply zigzag inverse
imag_inv=zeros(size(Imag));
for i=1:x/16
    for j=1:y/16
        block_num=16*(i 1)+j;
        sequens=quantized((block_num 1)*(Nc)+1:block_num*(Nc));
        templet=[sequens,zeros(1,N1)];

        block_inv=zigzag_inv(templet);
        imag_inv((i 1)*16+1:i*16,(j 1)*16+1:j*16)=block_inv;
    end
end

% apply dct inverse
dct_inv=zeros(size(Imag));
for i=1:x/16
    for j=1:y/16
        templet_1 = imag_inv((i 1)*16+1:i*16,(j 1)*16+1:j*16);
        dct_inv((i 1)*16+1:i*16,(j 1)*16+1:j*16)=idct2(templet_1);

    end
```

```
end


e=abs(Imag dct_inv);
PSNR = psnr(dct_inv,Imag);
MSSIM = ssim(dct_inv,Imag);

%%
figure()
set(gcf, 'Position', [100, 100, 1420, 960])
subplot(1,2,1)
imshow(Imag)
title("Original image from source",'FontSize',16,'FontWeight','bold')
subplot(1,2,2)
imshow(dct_inv)
title("Reconstructed image at receiver",'FontSize',16,'FontWeight','bold')
```

```
function [index,quantized,codebook]=quantization(signal)


left_value=min(signal);
right_value=max(signal);


partition=linspace(left_value,right_value,257);  % Decision threshold vector
codebook=linspace(left_value,right_value,256);   %Reconstruction levels

[index,quantized]=quantiz(signal,partition(2:end 1),codebook);   %scalar quantization
codebook=linspace(left_value,right_value,256);

end
```

```
function packets = packetizer(index,k)

row_number=ceil(size(index,2)/k);

packets=zeros(row_number,k);

    for i=1:row_number
        if i*k<=size(index,2)
            packets(i,:)=index((i 1)*k+1:i*k);
        else
            packets(i,1:size(index,2) (i 1)*k)=index((i 1)*k+1:end);
        end

    end
end
```

```
function [codes msgwords codewords] = rs_code(packets,m,k)

n=2^m 1;                        %codeword length
msgwords=gf(packets,m);         %represent packets using glaois array

codes=rsenc(msgwords, n, k);    %RS encoding
msgwords;
codewords=codes.x;              %Extract rows of codewords from GF array

end
```

```
function intrleaver = interlever(codes)
data=reshape(codes.',1,[]);

[n_row,n_colum]=size(codes);
```

```
intrlvd=matintrlv(data,n_row,n_colum);

intrleaver=reshape(intrlvd,n_colum,n_row);
intrleaver=intrleaver.';

end
```

```
function deintr = deinterleaver(codes_noisy,nw,nl)

data=reshape(codes_noisy.',1,[]);
inv_interleaver=matdeintrlv(data,nw,nl);
deintr=reshape(inv_interleaver,nl,nw);
deintr=deintr.';
end
```

```
function depackets = depacketizer(packets,index)

temp=[];                                    % empty templet matrix
depackets = reshape(packets.',1,[]);    % Resize the packets to fill the
    matrix
depackets = depackets(1:length(index));  % depacketize the signal

end
```

```
function scan_vector=zigzag(blocks)

[x, y]=size(blocks);
scan_vector=zeros(1,y*y);
scan_vector(1)=blocks(1,1);
a=1;
for k=1:2*y 1

    if k<=y
        if mod(k,2)==0
            f=k;
            for h=1:k
                scan_vector(a)=blocks(h,f);
                a=a+1;
                f=f 1;
            end
        else
            h=k;
            for f=1:k
                scan_vector(a)=blocks(h,f);
                a=a+1;
                h=h 1;
            end
        end
```

```
    else
        if mod(k,2)==0
            p=mod(k,y);
            f=y;
            for h=p+1:y
                scan_vector(a)=blocks(h,f);
                a=a+1;
                f=f 1;
            end
        else
            p=mod(k,y);
            h=y;
            for f=p+1:y
                scan_vector(a)=blocks(h,f);
                a=a+1;
                h=h 1;
            end
        end
    end
end

end
```

```
function scan_vector=zigzag_inv(templet)
% inverse transform from the zigzag format to the matrix form


N=sqrt(length(templet));
scan_vector=zeros(N,N);
scan_vector(1,1)=templet(1);
a=1;
for k=1:2*N 1

    if k<=N
        if mod(k,2)==0
            f=k;
            for h=1:k
                scan_vector(h,f)=templet(a);
                a=a+1;
                f=f 1;
            end
        else
            h=k;
            for f=1:k
                scan_vector(h,f)=templet(a);
                a=a+1;
                h=h 1;
            end
```

```
            end
        else
            if mod(k,2)==0
                p=mod(k,N);
                f=N;
                for h=p+1:N
                    scan_vector(h,f)=templet(a);
                    a=a+1;
                    f=f 1;
                end
            else
                p=mod(k,N);
                h=N;
                for f=p+1:N
                    scan_vector(h,f)=templet(a);
                    a=a+1;
                    h=h 1;
                end
            end
        end
    end

end
```