Wireless Receiver: 1 Mapping and Demapping

308681

Hiroki Hayakawa

**Task Part 1-1**

By reducing the number of bits to 1000, execution time decrease to about tenth (eg. 2.5sec to 0.25sec)

**Task Part 1-2**

Instead of mapping bits [0 1] to symbol A,B, we map the signal bits [0 1] to [-1 1]. This way makes it possible to apply BSC with plain Matrices/vectors operations and increase computing speed 20x faster (e.g. 2.5sec to 0.1sec).

Step1(Mapping): Map bits[0 1] to [-1 1], f(**x**) = 2\***x** -1

Step2(Channel): Generate vector whose components have random value in [0,1], and the transform the vector with following condition:

- If the value of the components is larger than 0.2, the value is 1.

- If the value of the components is smaller than 0.2, the value is -1.

By multiplying corresponding elements of the transformed vector and mapped bits, the elements of mapped bits are flipped.

Step3(Demapping): Demap [-1 1] to [1 1], g(**y**) = (**y**+1)/2

Step4(Error Count): Count error by taking difference between original bits and demapped one.

MATLAB code is as follows.

% Start time measurement

tic();

% Source: Generate random bits

txbits = randi([0 1],1000000,1);

% Mapping: [0 1] to [-1 1]

txbits\_MP = 2\*txbits - 1;

% Channel: Apply BSC

randval = rand(1000000,1) - ones(1000000,1)\*0.2;

randval = ceil(randval);

randval = randval\*2 - ones(1000000,1);

% multiplying corresponing elements of randval and Mapped bits, if rand<0.2 corresponding bits will be switched

rxbits = randval.\* txbits\_MP;

rxbits2 = (rxbits + ones(1000000,1) )\*0.5;

%error count

error = nnz(txbits - rxbits2);

% % Output result

disp(['BER: ' num2str(error/1000000\*100) '%'])

% Stop time measurement

toc()

**Task Part 2-1**

Following images are the received image with the function image\_decoder(revised version).



Fig. image generated by image\_decoder(revised version) with SNR of 10.



Fig. image generated by image\_decoder(revised version) with SNR of 1.

Matlab code of revised image\_decoder is as follows.

function image\_decoder2(b, image\_size)

% IMAGE\_DECODER - Decodes a bit stream into an image with

% a given fixed size and display it.

% image\_decoder(b, image\_size)

%

% Arguments:

% b: (vector) bit stream of length=8\*height\*width

% image\_size: (vector) [height, width] size of the image

%

% Author(s): unknown

% Copyright (c) 2011 RWTH.

% Convert the bit stream "b" into an image and display it.

% The vector "image\_size" of length 2 contains the dimensions

% (in pixels) of the image (height x width)

% Add AWGN to signal

SNR=1;

b = awgn(b,SNR);

% (Re,Im) --> [-1,1]

% (x1,x2), x1=cos(theta)/|cos(theta)|, x2=sin(theta)/|sin(theta)|

% theta in (0,pi/2) --> 11

% theta in (pi/2,pi) --> -11

% theta in (pi,3/2pi) --> -1-1

% theta in (3/2pi,2pi) --> 1-1

b2 = [cos(angle(b))./abs(cos(angle(b))),sin(angle(b))./abs(sin(angle(b)))];

%[11,-11,-1-1,1-1] --> [11,01,00,10]

b3 = (b2+ones(length(b2),2))\*0.5;

% error handling

if numel(b3) ~= 8 \* prod(image\_size)

error('Input vector has wrong size.')

end

% convert to uint8

b4 = transpose(b3);

b5 = reshape(b4,numel(b4),1);

b6= reshape(b5, 8, length(b5)/8).';

image = bi2de(b6);

% reshape into image format

image = reshape(image, image\_size(2), image\_size(1)).';

% displax image

imageview(image);

return

**Task Part 2-2 ~ 2-4**

Comparing BER plots generated from the two mapping schemes, we found the mapping scheme shown in Fig 1.4 generates smaller BER. In the mapping scheme shown in Fig 1.6, value of 1st bit is used to decode 2nd bid, which means noise on the 1st bit affect the 2nd bit and worsen the error rate of the decoded signal.

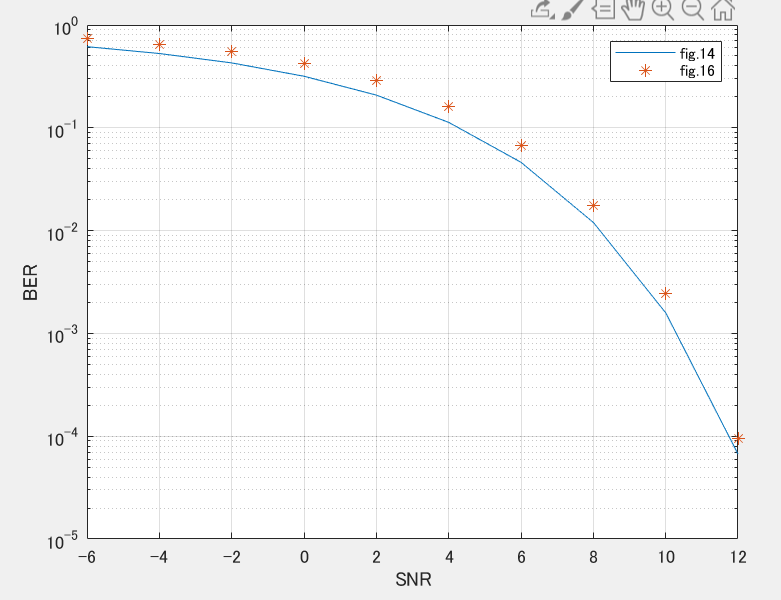


Fig. BER plot (mapping scheme in fig 1.4 and fig.1.6)

Matlab code for task part2-2~2-4 is as follows.

load image.mat

num\_data = 1000000;

% Source: Generate random bits

txbits = randi([0 1],num\_data,2);

% Call function to calculate BER (Fig.14)

[PL,QPSK] = BER\_Calc(txbits,num\_data);

% Call function to calculate BER (Fig.16)

[PL2,QPSK] = BER\_Calc2(txbits,num\_data);

% Comparing plots of noisy constellations

figure(1)

plot(txbits(:,1),txbits(:,2),"o");

hold on

plot(QPSK,"\*");

hold off

% log scale plot SNR-BER

figure(2)

semilogy(PL(:,1),PL(:,2));

hold on

semilogy(PL2(:,1),PL2(:,2),'\*');

hold off

grid on

xlabel('SNR')

ylabel('BER')

legend('fig.14','fig.16')

\*function to calculate BER (QPSK in fig 1.4)

function [PL,QPSK]=BER\_Calc(txbits, num\_data)

% Mapping to QPSK symbols(in fig 1.4)

% [11,01,00,10] -->[11,-11,-1-1,1-1]

txbits\_2 = txbits\*2 - ones(num\_data,2);

QPSK = complex(txbits\_2(:,1),txbits\_2(:,2))/sqrt(2);

SNR=-6;

i = 1;

while SNR <=12

% Add AWGN to QPSK signal

Noised\_Signal = awgn(QPSK,SNR);

% Noised signal to QPSK (Angle = theta)

% (x1,x2), x1=cos(theta)/|cos(theta)|, x2=sin(theta)/|sin(theta)|

% theta in (0,pi/2) --> 11

% theta in (pi/2,pi) --> -11

% theta in (pi,3/2pi) --> -1-1

% theta in (3/2pi,2pi) --> 1-1

QPSK\_WN = [cos(angle(Noised\_Signal))./abs(cos(angle(Noised\_Signal))),sin(angle(Noised\_Signal))./abs(sin(angle(Noised\_Signal)))];

% QPSK signal to bits: [11,-11,-1-1,1-1] --> [11,01,00,10]

rxbits = (QPSK\_WN+ones(num\_data,2))\*0.5;

% Error count

error = nnz(txbits - rxbits);

BER = error/num\_data;

% Record SNR and Correspondet BER

PL(i,1)=SNR;

PL(i,2)=BER;

% Repeat with other SNR

SNR = SNR + 2;

i = i + 1;

end

end

\*function to calculate BER (QPSK in fig 1.6)

function [PL,QPSK]=BER\_Calc2(txbits, num\_data)

% Mapping to QPSK symbols(in fig 1.6)

% [01,10,11,00] -->[11,-11,-1-1,1-1]

txbits\_1 = txbits(:,1)\*-2 + ones(num\_data,1);

txbits\_2 = 2\*txbits\_1.\*txbits(:,2)-txbits\_1.\*ones(num\_data,1);

QPSK = complex(txbits\_1,txbits\_2)/sqrt(2);

SNR=-6;

i = 1;

while SNR <=12

% Add AWGN to QPSK signal

Noised\_Signal = awgn(QPSK,SNR);

% Noised signal to QPSK (Angle = theta)

% (x1,x2), x1=cos(theta)/|cos(theta)|, x2=sin(theta)/|sin(theta)|

% theta in (0,pi/2) --> 11

% theta in (pi/2,pi) --> -11

% theta in (pi,3/2pi) --> -1-1

% theta in (3/2pi,2pi) --> 1-1

QPSK\_WN = [cos(angle(Noised\_Signal))./abs(cos(angle(Noised\_Signal))),sin(angle(Noised\_Signal))./abs(sin(angle(Noised\_Signal)))];

% QPSK signal to bits [11,-11,-1-1,1-1] --> [01,10,11,00]

rxbits\_1 = (QPSK\_WN(:,1)-ones(num\_data,1))\*-0.5;

rxbits\_2 = 0.5\*QPSK\_WN(:,1).\*(QPSK\_WN(:,1)+QPSK\_WN(:,2));

rxbits = [rxbits\_1, rxbits\_2];

% Error count

error = nnz(txbits - rxbits);

BER = error/num\_data;

% Record SNR and Correspondet BER

PL(i,1)=SNR;

PL(i,2)=BER;

% Repeat with other SNR

SNR = SNR + 2;

i = i + 1;

end

end

**Task 2-5**

Debagged code is as follows.

function compressed\_decoder2(b,image\_size)

% (Re,Im) --> [-1,1]

% (x1,x2), x1=cos(theta)/|cos(theta)|, x2=sin(theta)/|sin(theta)|

% theta in (0,pi/2) --> 11

% theta in (pi/2,pi) --> -11

% theta in (pi,3/2pi) --> -1-1

% theta in (3/2pi,2pi) --> 1-1

b2 = [cos(angle(b))./abs(cos(angle(b))),sin(angle(b))./abs(sin(angle(b)))];

%[11,-11,-1-1,1-1] --> [11,01,00,10]

b3 = (b2+ones(length(b2),2))\*0.5;

% error handling

if numel(b3) ~= 8 \* prod(image\_size)

error('Input vector has wrong size.')

end

% % convert to uint8

% b1 = reshape(b, 8, length(b)/8).';

% image = bi2de(b1);

% convert to uint8

b4 = transpose(b3);

b5 = reshape(b4,numel(b4),1);

b6= reshape(b5, 8, length(b5)/8).';

image = bi2de(b6);

size(image);

% reshape into image format

image = reshape(image, image\_size(2), image\_size(1)).';

size(image);

% pad the image to multiples of 8

height = ceil(image\_size(1)/8)\*8;

width = ceil(image\_size(2)/8)\*8;

padded = zeros(height,width);

size(padded);

padded(1:image\_size(1),1:image\_size(2)) = image;

factor = 0.7;

% segment image into tiles

data = zeros(8\*8,(width\*height)/(8\*8));

k = 1;

for rr=1:8:width % go through rows

for cc=1:8:height % go through columns

patch = padded(cc:cc+7,rr:rr+7);

vector = reshape(patch,8\*8,1);

data(:,k) = vector;

k = k + 1;

end

end

% decompose

[U S V] = svd(data.',0);

% filter weak singular values

fullsv = diag(S);

compressedsv = zeros(size(fullsv));

mask = cumsum(fullsv) < (factor\*sum(fullsv));

compressedsv(mask) = fullsv(mask);

SC = diag(compressedsv);

% compose

newdata = U\*SC\*V';

size(newdata);

newpadded = zeros(height,width);

k = 1;

% assemble full image from patches

for cc=1:8:width % go through rows

for rr=1:8:height % go through columns

patch = newdata(k,:);

newpadded(rr:rr+7,cc:cc+7) = reshape(patch,8,8);

k=k+1;

end

end

% shrink to original size

newimage = newpadded(1:image\_size(1),1:image\_size(2));

imageview(newimage);

end

**Theory Task 3.1**

Each point in the red constellation has same distance from origin. This means each point has same energy and it is equal to the average energy.

P\_red = 3^2 + 3^2 =18

In the blue constellation, four points closer to the origin has same energy of 3^2+0=9, and remaining two points have same energy of 6^2+6^2=72. Thus the average energy of the blue constellation is caluculated as follows.

P\_blue = (9\*4 + 72\*2) / 6 = 30

**Theory Task 3.2**

Energy-normalized constellation points can be obtained by dividing original constellation points by their average energy. Following figure shows energy-normalized constellation points of the two schemes(red and blue).

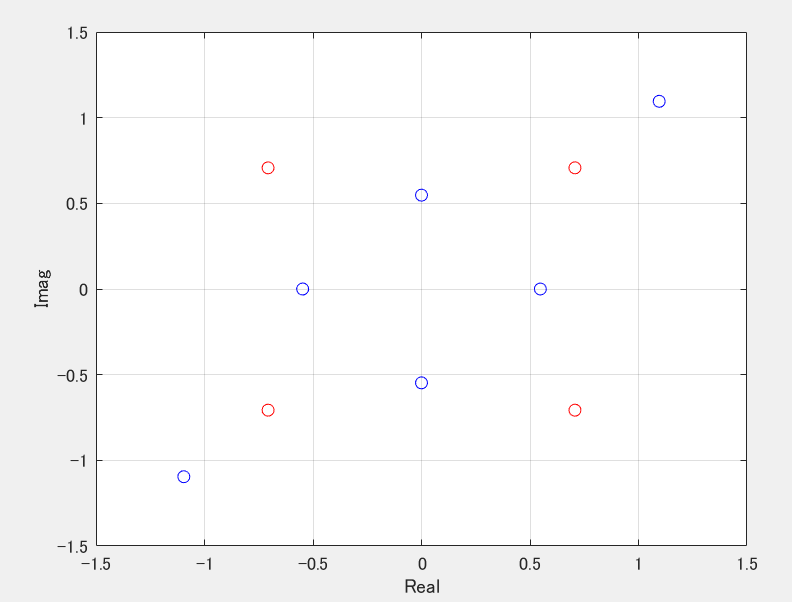


Fig. energy-normalized constellation points of the two schemes(red and blue)