Project #1 Modern Coding Theory Spring, 2022

Topic : Performance of Turbo codes based on the BCJR algorithm

20212145 김 희서

1. introduction

[1]4, 5G 기술이 확립된 이후, 태블릿 PC, 스마트워치 등이 널리 보급되어 많은 기기들이 무선 통신 네트워크에 참여하고 있다.

이런 사용자들의 수요와 함께 증가한 데이터 요구량과 고품질 서비스를 구현할 수 있는 현재 5, 6G 기술을 활발히 연구하고 있다.

특히 채널 코딩은 통신 채널에서 발생할 수 있는 에러를 수정하는 기술로, 최근 높은 오류정정 능력과 빠른 인코딩, 디코딩을 통해, 5G 통신 시스템에서 요구하는 사항인 high data rate, low latency, high reliability를 달성할 수 있도록 연구가 진행 중이다.

해당 프로젝트는 5G시스템에서 적용 중인 LDPC, polar 코드를 학습하기 전에 공부한 turbo 코드의 interleaver size, iteration 수에 따른 시뮬레이션을 진행하였습니다.

2. turbo code

Turbo 코드는 현재 4G LTE 시스템을 비롯하여 널리 사용된 코드로, Turbo 코드의 encoder는 여러 개의 component encoder와 interleaver로 구성되어 있다. 각 입력 정보 비트의 사후 확률을 계산하여 입력 정보 비트의 오류가 최소가 되도록 decoding 한다. 이 알고리즘은 decoder에서 소프트 출력을 구해야하는 turbo 코드에서 사용되면서 주목받았다.

Turbo 코드는 convolutional 코드로 구성되어 있기에, convolutional decoder를 이용하여, 디코딩을 수행 한다.

이때 디코딩 기법으로는 여러 가지가 존재하지만, BCJR decoding을 일반적으로 사용한다. 이 BCJR 디코딩은 convolutional 코드의 maximum a posteriori(MAP) decoder로 알려져 있고, turbo 코드의 확률적 반복 디코딩을 위한 soft-input soft-output(SIS)) 디코딩이 가능하다.

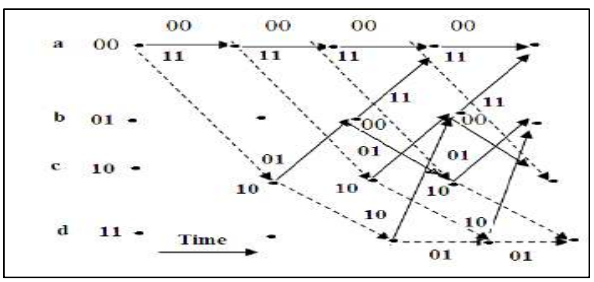


그림 1 Trellis diagram for convolution code

그림 2는 feedback이 존재하고, rate가 1/2인 RSC 인코더 두 개로 구성되어 있다. 이때 interleaver는 시뮬레이션 코드에서 “intr_map = randperm(num_bit);“로 구현한 것처럼, random interleaver로 구성되어 있다.

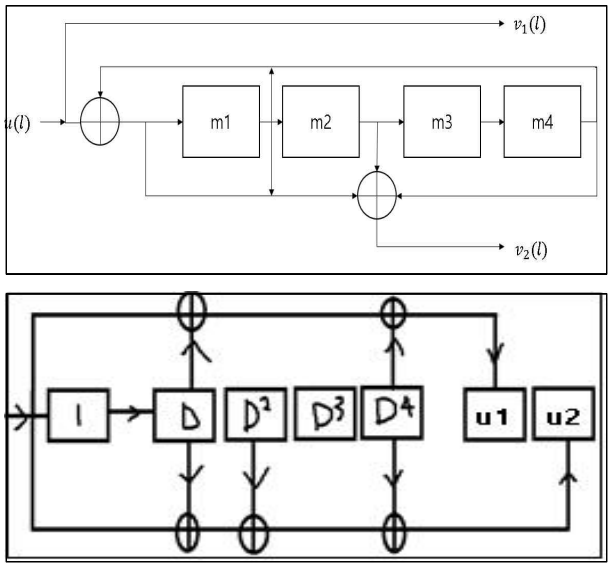


그림 2 $G(D)=[1\ \frac{1+D+D^2+D^4}{1+D+D^4}]$

input	state				output	
0	0	0	0	0	0	0
	0	0	0	1	1	1
	0	0	1	0	0	0
	0	0	1	1	1	1
	0	1	0	0	0	1
	0	1	0	1	1	0
	0	1	1	0	0	1
	0	1	1	1	1	0
	1	0	0	0	1	1
	1	0	0	1	0	0
	1	0	1	0	1	1
	1	0	1	1	0	0
	1	1	0	0	1	0
	1	1	0	1	0	1
	1	1	1	0	1	0
	1	1	1	1	0	1

input	state				output	
1	0	0	0	0	1	1
	0	0	0	1	0	0
	0	0	1	0	1	1
	0	0	1	1	0	0
	0	1	0	0	1	0
	0	1	0	1	0	1
	0	1	1	0	1	0
	0	1	1	1	0	1
	1	0	0	0	0	0
	1	0	0	1	1	1
	1	0	1	0	0	0
	1	0	1	1	1	1
	1	1	0	0	0	1
	1	1	0	1	1	0
	1	1	1	0	0	1
	1	1	1	1	1	0

3. 시뮬레이션

해당 시뮬레이션의 주제는 “Performance of Turbo codes based on the BCJR algorithm”이다.

주어진 조건은 다음과 같다.

① AWGN channels are assumed.

② The BCJR algorithm is used for decoding.

③ The number of iterations is limited by 6, 8, 10, 15, 20.

④ Random interleavers are assumed.

⑤ The interleaver size are 200, 500 and 1000.

⑥ The RSC code as a component code has one of the following transfer matrices.

① AWGN channels are assumed.

```
%---------------- SIMULATION PARAMETERS ------------------
SNR_dB = 0.5; % SNR per bit in dB (in logarithmic scale)
sim_runs = 1*(10^3); % simulation runs
frame_size = 2000; % frame size
num_bit = 0.5*frame_size; % number of data bits
SNR = 10^(0.1*SNR_dB); % SNR per bit in linear scale
noise_var_1D = 2*2/(2*SNR); % 1D noise variance
```

“SNR_dB”을 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5로 설정.

⑤ The interleaver size are 200, 500 and 1000.)

“frame_size”를 바꿔가면서, interleaver size를 조정하였다.

“num_bit”에 0.5를 통해, rate가 1/2임을 알 수 있다.

```
% AWGN
white_noise = sqrt(noise_var_1D)*randn(1,frame_size)+1i*sqrt(noise_var_1D)*randn(1,frame_size);
Chan_Op = mod_sig + white_noise; % Chan_Op stands for channel output

SNR_dB1 = 1; % SNR per bit in dB (in logarithmic scale)
SNR1 = 10^(0.1*SNR_dB1); % SNR per bit in linear scale
noise_var_1D1 = 2*2/(2*SNR1); % 1D noise variance
white_noise1 = sqrt(noise_var_1D1)*randn(1,frame_size)+1i*sqrt(noise_var_1D1)*randn(1,frame_size);
Chan_Op1 =  mod_sig + white_noise1;

SNR_dB2 = 1.5; % SNR per bit in dB (in logarithmic scale)
SNR2 = 10^(0.1*SNR_dB2); % SNR per bit in linear scale
noise_var_1D2 = 2*2/(2*SNR2); % 1D noise variance
white_noise2 = sqrt(noise_var_1D2)*randn(1,frame_size)+1i*sqrt(noise_var_1D2)*randn(1,frame_size);
Chan_Op2 = mod_sig + white_noise2;
```

그 값에 따라 각 해당하는 “noise_var_1D”값을 설정하였고, “Chan_op”는 channel output을 나타낸다.

④ Random interleavers are assumed.

```
%    Generator polynomial of the component encoders
gen_poly = ldiv2([1 1 1 0 1],[1 1 0 0 1],num_bit); % using long division method

%  Interleaver and deinterleaver mapping of the turbo code
intr_map = randperm(num_bit);
deintr_map = deintrlv((1:num_bit),intr_map);
```

gen_poly를 통해, $G(D)=[1\ \frac{1+D+D^2+D^4}{1+D+D^4}]$를 나타내었고, randperm을 통해, “Random interleavers”를 구현하였다.

② The BCJR algorithm is used for decoding.

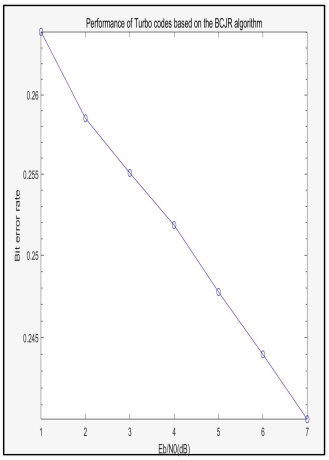
③ The number of iterations is limited by 6, 8, 10, 15, 20.

```
% iterative logMAP decoding
LLR = log_BCJR(LLR,log_gamma1,num_bit); % outputs extrinsic information
LLR = log_BCJR(LLR(intr_map),log_gamma2,num_bit); %1

LLR = log_BCJR(LLR(deintr_map),log_gamma1,num_bit);
LLR = log_BCJR(LLR(intr_map),log_gamma2,num_bit); %2

LLR = log_BCJR(LLR(deintr_map),log_gamma1,num_bit);
LLR = log_BCJR(LLR(intr_map),log_gamma2,num_bit); %3

LLR = log_BCJR(LLR(deintr_map),log_gamma1,num_bit);
LLR = log_BCJR(LLR(intr_map),log_gamma2,num_bit); %4

 LLR = log_BCJR(LLR(deintr_map),log_gamma1,num_bit);
 LLR = log_BCJR(LLR(intr_map),log_gamma2,num_bit); %5

 LLR = log_BCJR(LLR(deintr_map),log_gamma1,num_bit);
 LLR = log_BCJR(LLR(intr_map),log_gamma2,num_bit); %6
```

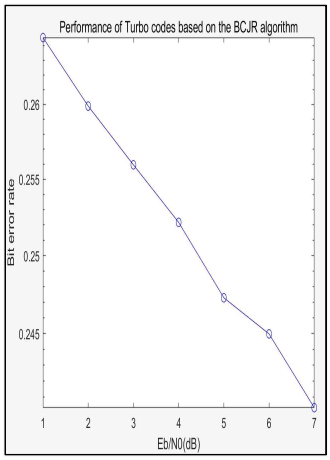
다음의 코드에서 BCJR 알고리즘과 interation에 대한 조건을 구현하였다.

3.1 iteration

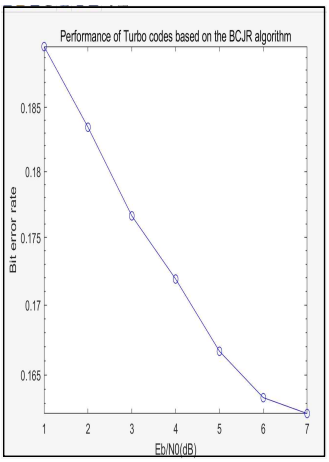
A) interleaver size : 200



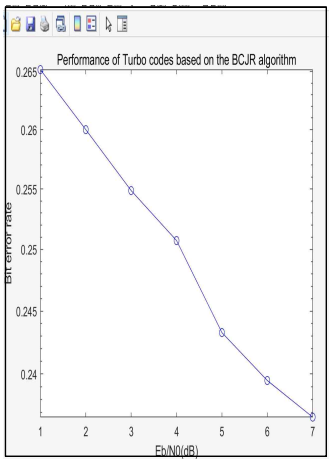
number of iterations : 4



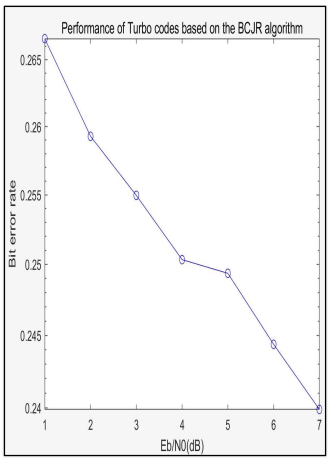
number of iterations : 6



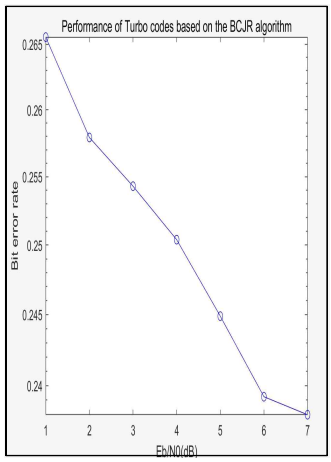
number of iterations : 8



number of iterations : 10



number of iterations : 15



number of iterations : 20

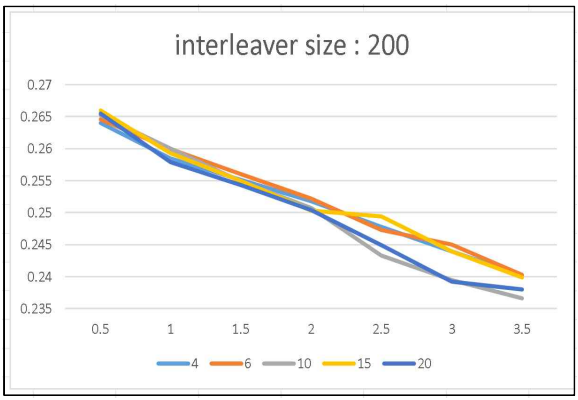


그림 4 interleaver size : 200

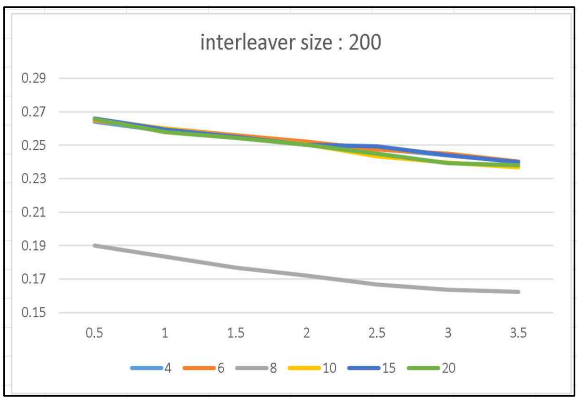


그림 5 interleaver size : 200

다음 시뮬레이션 결과 값은 interleaver size가 200일 때의 값으로, SNR_dB값이 0.5, 1, 1.5, 2.0, 2.5, 3.0, 3.5일 때의 BER 값을 출력하였다.

시뮬레이션의 공통적인 특징은 iteration 8일 때의 값이 튀는 값이 나왔기에, iteration에 따른 성능을 보기에 적합하지 않다고 생각하여, 그림4와 같이 iteration 8일 때를 제외한 경우의 그래프를 출력하였다.

iteration에 따른 결과 값을 비교하기 위해, iteration의 수가 4일 때와 20일 때의 값을 비교하면, 0.5, 1dB에서는 BER의 값의 차이가 적지만, 2.5, 3.dB에서는 각각의 BER 값의 차이가 전보다 커진 것을 볼 수 있다. 따라서 iteration에 따른 성능은 dB값이 클수록 개선되는 것을 알 수 있다.

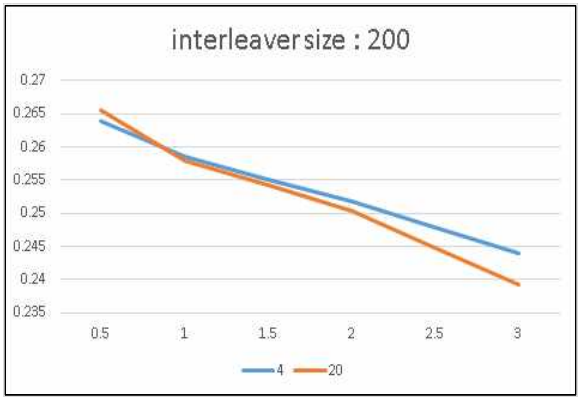
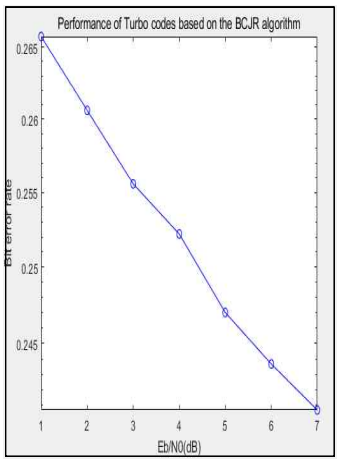


그림 6 interleaver size : 200

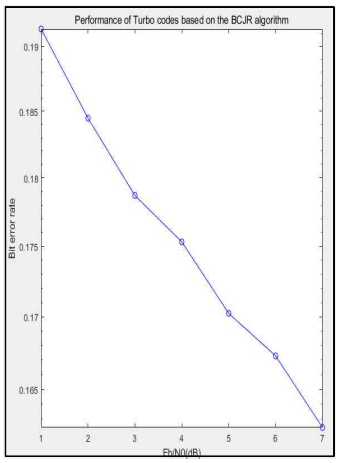
즉, 위의 그림 6처럼 3dB에서 iteration이 4, 20일 때 BER 값의 차이가 큰 것을 확인 할 수 있다.

이를 통해, iteration이 증가함에 따라 성능이 개선되어, BER의 값이 변하는 것을 확인하였다.

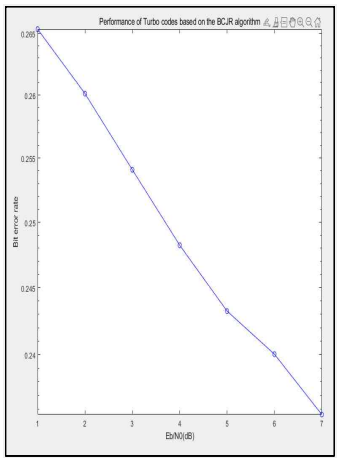
B) interleaver size : 500



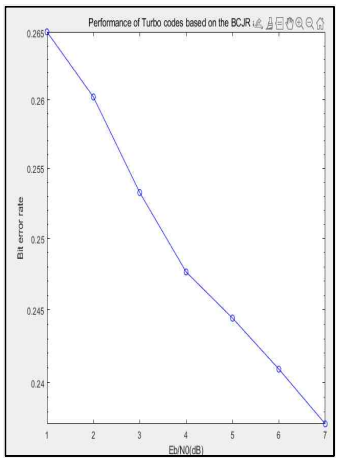
number of iterations : 6



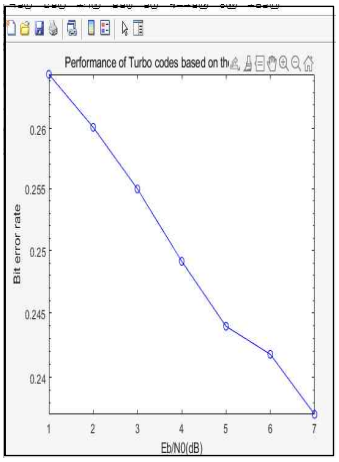
number of iterations : 8



number of iterations : 10



number of iterations : 15



number of iterations : 20

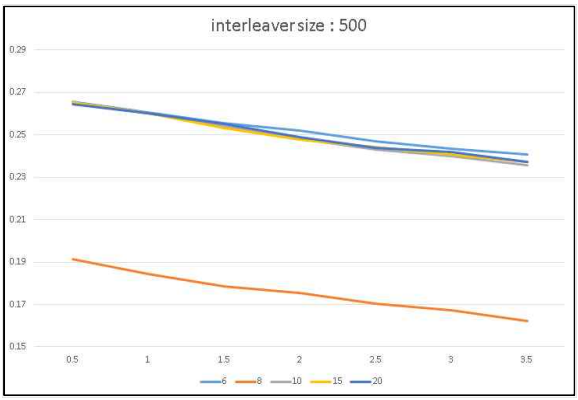


그림 7 interleaver size : 500

interleaver size가 500일 때도 iteration 8의 값이 튀는 값이 나왔고, iteration에 따른 결과 값을 확인하기 위해서 그림 8처럼 iteration 6, 20일 때의 결과 값을 비교하였다.

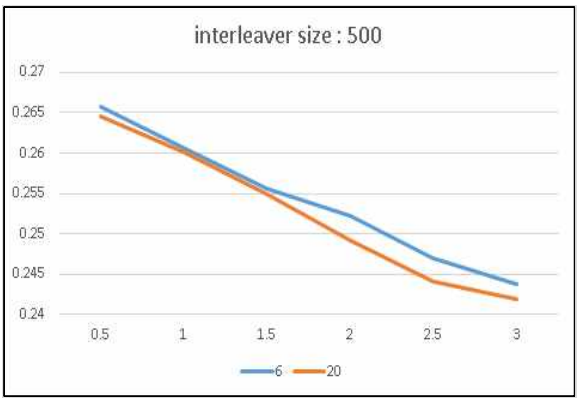
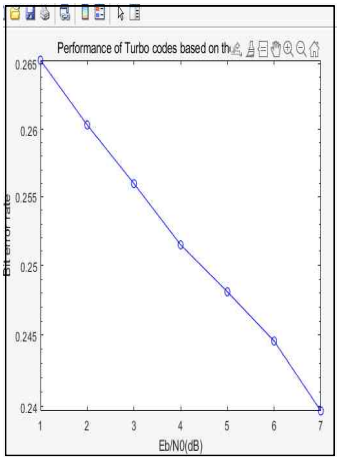


그림 8 interleaver size : 500

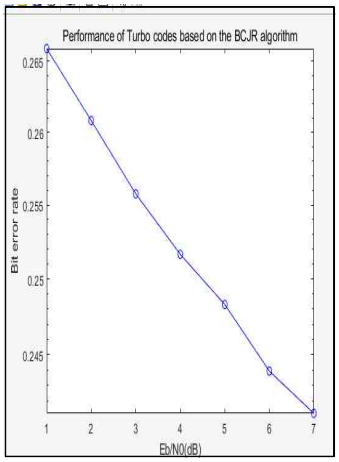
200일 때처럼 iteration 6과 20일 때 0.5dB일 때 보다, 2, 2.5dB로 커질수록, BER값의 차이가 커지는 것을 확인할 수 있다.

이를 통해, iteration이 증가함에 따라 성능이 개선되어, BER의 값이 변하는 것을 확인하였다.

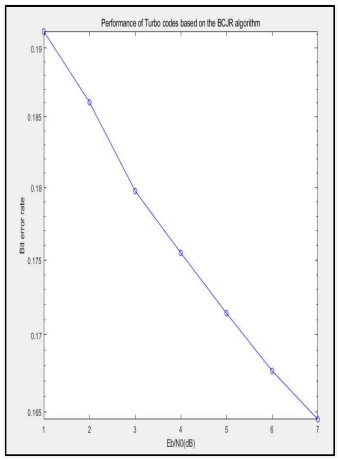
C) interleaver size : 1000



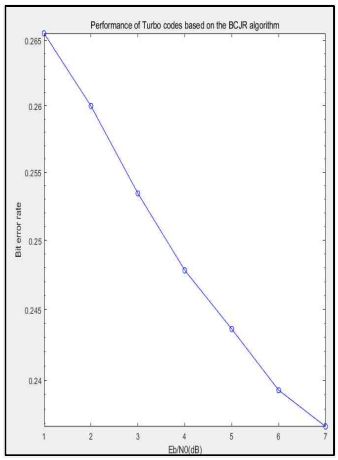
number of iterations : 4



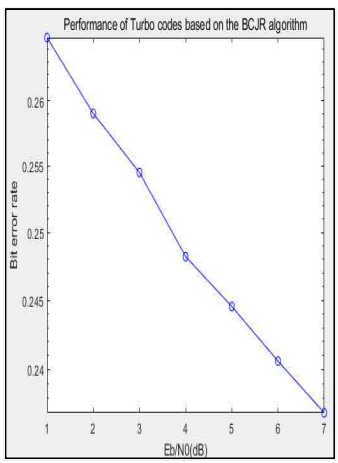
number of iterations : 6



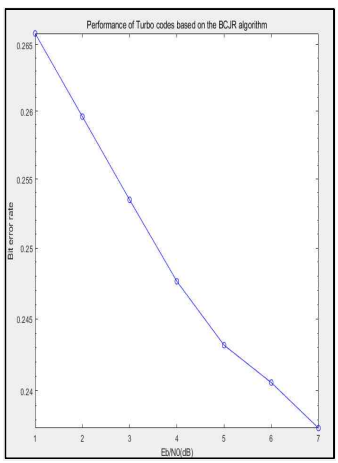
number of iterations : 8



number of iterations : 10



number of iterations : 15



number of iterations : 20

다음은 interleaver size가 1000일 때의 값을 시뮬레이션 한 것이다.

이때도 이전의 interleaver size가 200, 500일 때와 같은 결과가 나왔고, 그림10에서도 iteration 6과 20일 때의 BER 값의 차이를 통해, iteration에 따라 성능이 개선되는 것을 확인할 수 있다.

그림 9 interleaver size : 1000

그림 10 interleaver size : 1000

지금까지 interleaver size를 200, 500, 1000으로 고정하고, iteration에 따른 시뮬레이션 값을 확인하였다.

[2]결과적으로 iteration이 증가함에 따라 BER값이 0.5, 1dB와 같은 작은 값보단 2.5, 3dB와 같이 큰 값에서 성능이 개선되는 것을 확인할 수 있었다.

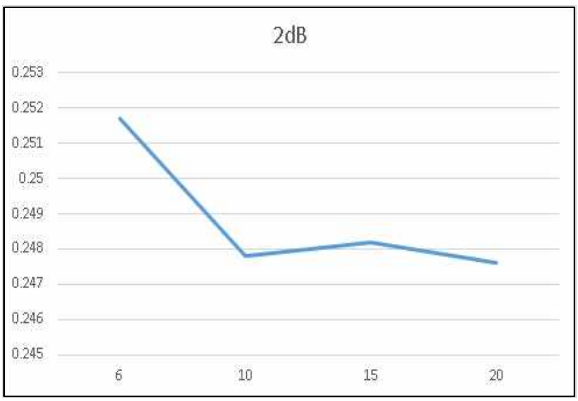


그림 11 interleaver size : 1000

다음은 Iteration 수를 “6, 10, 15, 20”으로 바꿨을 때, 인터리버의 성능을 본 것이다.

이때 시뮬레이션에 사용한 랜덤 인터리버의 iteration 수가 증가할수록 BER 감소가 되는데, 이는 랜덤 인터리버의 불규칙성이 크므로 비트 재배치로 인한 성능이 증가하기 때문이다. iteration이 10일 때, 급격한 BER 값이 감소되지만, 그 이후로는 큰 변화가 없다는 것을 확인할 수 있다.

3.2 interleaver size

이번 시뮬레이션에서는 같은 iteration일 때, interleaver size에 따른 결과 값을 비교하였다.

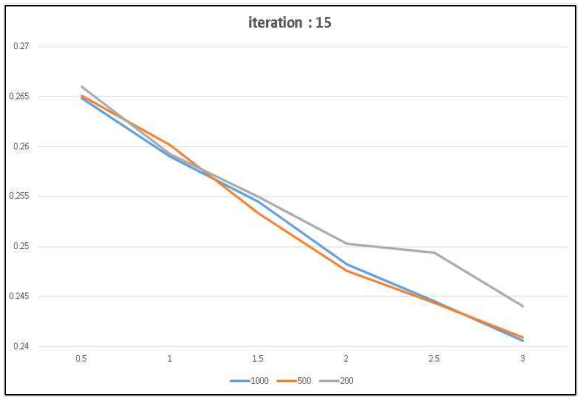


그림 11 iteration : 15

iteration 15일 때의 interleaver size에 따른 시뮬레이션 값을 비교하였다. size가 클수록 작은 BER을 갖는 것을 확인할 수 있다.

4. 결론

해당 시뮬레이션은 BCJR 알고리즘 기반으로 한 터보 코드의 성능을 비교한 것이다.

iteration이 증가함에 따라, interleaver size가 200, 500, 1000일 때의 BER값에서의 성능 개선되는 것을 확인할 수 있다.

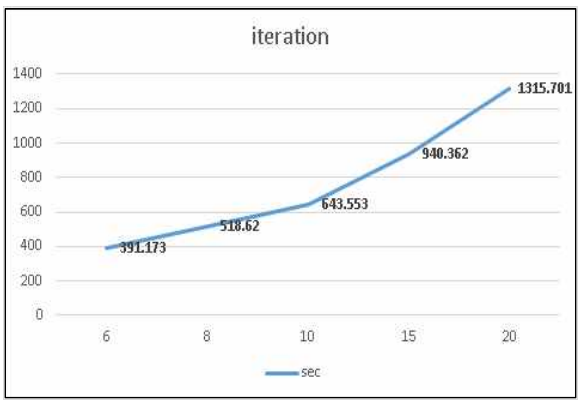


그림 12 interleaver size : 500

성능은 개선되지만, 시뮬레이션 동작시간이 오래 걸리는 것을 확인할 수 있었다.

그리고 같은 iteration일 때, 서로 다른 interleaver size를 비교하여, 시뮬레이션을 진행하였는데, 이를 통해 interleaver size가 클 때 BER값을 통해 성능이 개선되

는 것을 확인할 수 있었다.

해당 시뮬레이션을 통해, 터보코드에 대한 이해를 높일 수 있었지만, 생각보다 극적인 변화가 많이 없었고, iteration 8일 때, 공통적으로 튀는 값이 발생하였다.

그 원인으로는 trellis 구현하는 코드를 수정하는 과정에서 발생한 것으로 생각하고, 주어진 조건인 G(D)에 맞춰, trellis 구현 코드를 완전히 구현하지 못한 것이 가장 큰 시뮬레이션에서 오차가 발생한 이유라고 생각한다.

참고문헌

[1] 박진수, 김인선, “5G를 위한 채널 코드 후보 기술과 연구 동향”, The Proceedings of the Korean Institute of Electromagnetic Engineering and Science v.27 no.6 , 2016년, pp.20 - 28.

[2] Prabhavati D. Bahirgonde, “BER Analysis of Turbo Decoding Algorithms”, International Journal of Computer Applications, vol. 126, No.14, Sep. 2015.

실행파일 : file.m
source file : Get_Trellis.m, ldiv2.m, log_BCJR.m, log_BCJR_END.m