**Digital Communications Term Project**

12181457 김범수

1. **Flow Chart**

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이번 Term Project는 Eb/No에 따른 BER을 구하고 Plot하여 이론적인 값과 비교하는 것으로 이루어져 있다.

이 과정에서 Modulation 방식과 Coding 방법에 따라 Curve가 어떻게 변화하는지 확인해보았다.

먼저 Message Bits에 해당하는 Random bits를 생성하였다.

생성한 Bit를 Modulation 방식에 따라 Modulation을 진행하고 각 채널 환경에 따라 에러 환경을 구성하였다.

Random noise와 Ransom Channel을 생성하고 이를 송신하면서 Message Bits가 변경되는 상황을 표현하도록 설정하였다.

이후 Equalizer을 통하면서 Channel에 대한 보상을 진행하고 Demodulation을 진행하였다.

이렇게 Demodulation 된 Receive Bits를 원본 Bits와 비교하여 Error를 계산하였다.

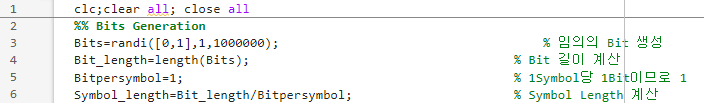
위 과정을 Eb/No을 변경하며 반복하였고 이를 통해 BER을 구하고 Eb/No을 구한 후 Plot 하였다.

위 과정에서 FEC(Forward Error Correction) 방법에 따라 Modulation을 진행하기 전에 Encoding 해 주었고 Demodulation 이후 Syndrome, Viterbi algorithm에 따라 Decoding을 진행했다.

1. **BPSK, QPSK, 16QAM BER vs Eb/No Curve Programming**

**2-1. BPSK Simulation (AWGN)**

**<Bits Generation>**



먼저 송수신에 사용할 Bits를 0과 1로 구성된 Random한 정수로 107개 생성해 주었다.

이는 Target BER인 10-5을 맞추기 위함이다.

100개 이상의 표본이 쌓이면 신뢰할 수 있다고 판단할 수 있기 때문에 107개로 설정하였다.

이후 Length()함수를 이용해서 생성된 Bits의 길이를 구하였다.

BPSK는 하나의 Symbol이 1개의 Bit를 표현하므로, Bit per Symbol은 1로 선언하였다.

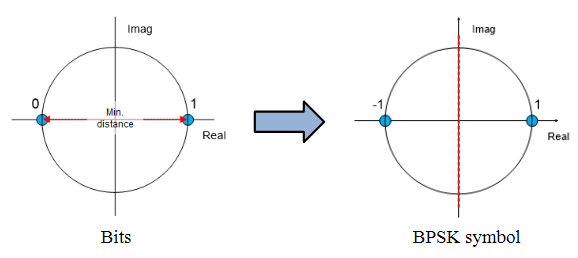
Symbol의 길이는 Bits를 Bit per Symbol로 나누면 되므로 이를 수식으로 작성하여 Symbol의 길이를 구하였다.

**<Modulation>**



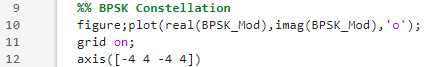
Modulation은 위 코드를 통하여 구현하였다.

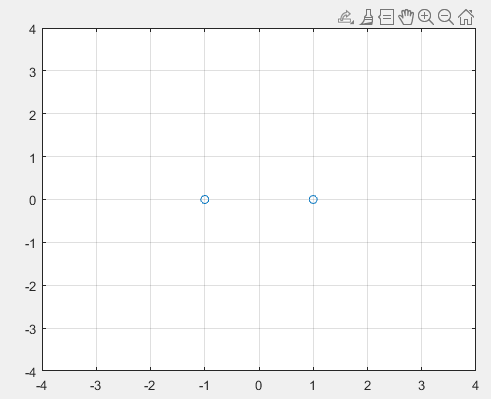
각각의 Bit가 0인 경우 -1로, 1인 경우 1로 Mapping을 진행하였다.



이 때 Modulation 값은 0 또는 1 값을 가져 별도의 Normalization은 진행하지 않았다.

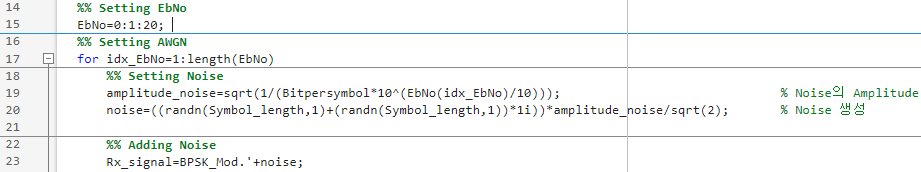
**<Constellation>**

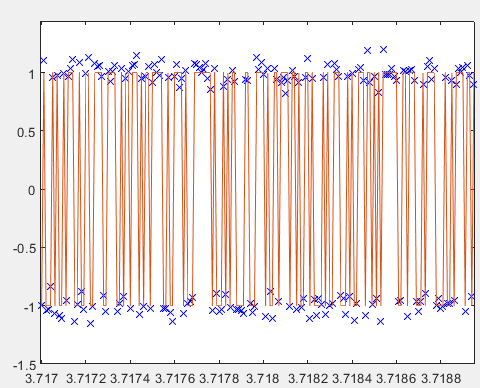
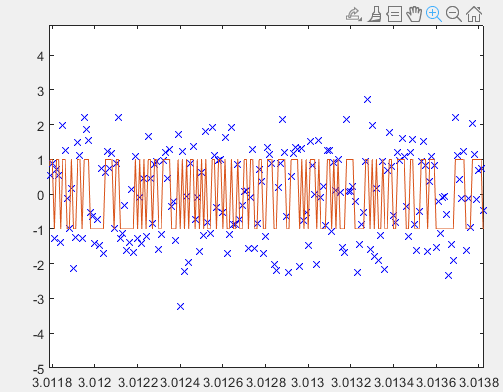




위 코드를 통해 만든 Constellation을 Plot하였다.

**<Setting AWGN>**



-Eb/No : 0-Eb/No : 20

위 코드는 EbNo을 0부터 20까지 1씩 증가하도록 생성 후 EbNo에 따라 다른 Noise를 생성하도록 설정하였다.

이때 Noise의 Amplitude는 아래와 같으며 Eb/No이 커짐에 따라 작아지는 형태를 가지고 있다.

위 수식에 따르면 Eb/No이 0인 경우 Noise의 Amplitude가 1이 되어 최댓값을 가지게 되고 Eb/No이 20인 경우 Noise의 Amplitude는 0.1이 되어 최솟값을 가지게 된다.

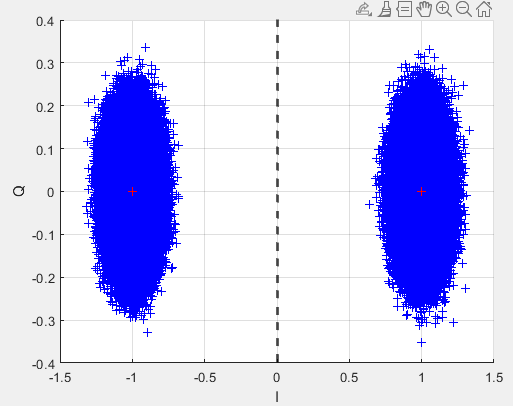
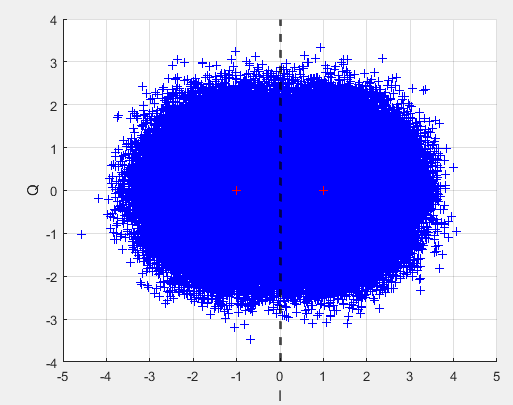
Noise를 수치로 표현하고 Modulation 된 값과 연산을 진행하기 위하여 Symbol의 길이만큼의 Noise 배열을 생성하고 root(2)로 nomalize를 진행하고 Amplitude를 곱하여 표현하였다.

위 Eb/No에 따른 그래프에서 볼 수 있듯 Eb/No이 0인 경우 빨간색 Modulation된 Bits에서 벗어난 파란색 x를 볼 수 있다.

이는 Noise의 Amplitude가 커지면서 Noise가 더해진 Signal이 큰 편차를 보이는 것으로 생각해볼 수 있다.

반대로 Eb/No이 20인 경우 Bits 값과 유사한 곳에 파란색 x 표시가 존재하는 것을 확인해 볼 수 있다.

이는 Noise의 Amplitude가 작아지면서 Noise가 더해진 Signal이 작은 편차를 보이는 것으로 생각해 볼 수 있다.

-Eb/No : 0-Eb/No : 20

위 결과는 Constellation에서 살펴보면 위와 같은 결과를 얻을 수 있다.

Decision Boundary를 살펴보면 Eb/No이 작은 쪽이 더 많은 영역을 침범하는 것을 확인할 수 있다.

따라서 Eb/No이 작은 경우 BER이 높을 것을 생각해볼 수 있다.

**<Demodulation>**



BPSK에서 Threshold는 1과 -1의 중심인 0을 기준으로 수신 신호를 분류한다.

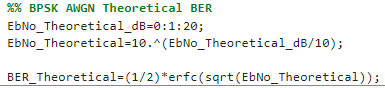
Demodulation 된 값을 저장하기 위하여 Bit length 만큼의 열을 가지는 행렬을 0으로 초기화하여 생성하고 0을 기준으로 0보다 크면 1을 대입하고 0보다 크지 않은 경우 0으로 그대로 저장된다.

**<BER>**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

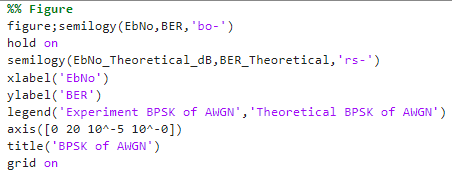
자동 생성된 설명

BER은 Receive Bits와 Send Bits를 비교하고 다른 개수를 더해서 저장한 후 Bits의 length로 나눠 비율을 구하였다.

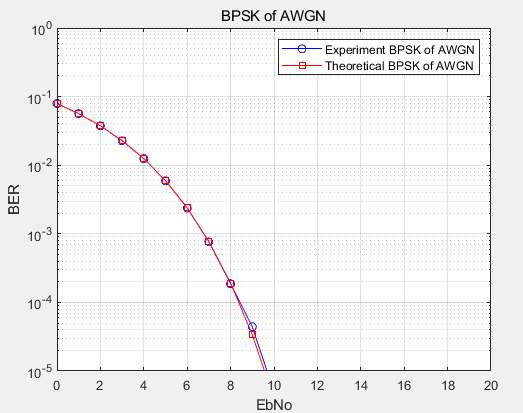


이후 이론적인 값과 비교하기 위하여 Eb/No을 설정하고 수식에 대입하여 값을 저장해주었다.

**<Figure>**



위 코드를 통하여 코드로 작성한 부분과 이론적인 부분을 비교하여 Plot하였다.

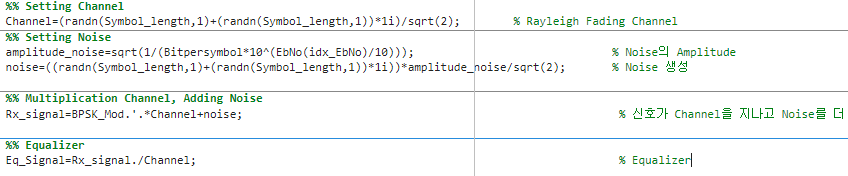


이론적인 내용과 매우 유사하게 나타난 것을 확인할 수 있었다.

약간의 차이를 보이는 부분은 Bits의 길이를 더 길게 하면 줄어들 것으로 생각된다.

**2-2. BPSK Simulation (Rayleigh Fading)**

**<Setting Channel & Noise>**

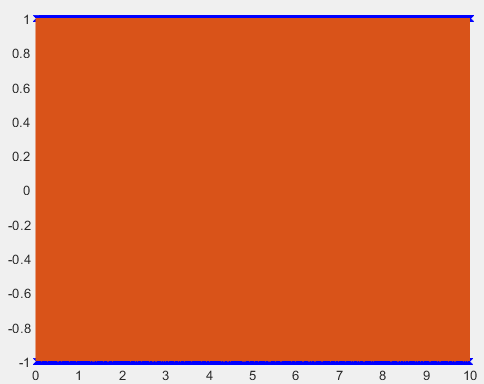
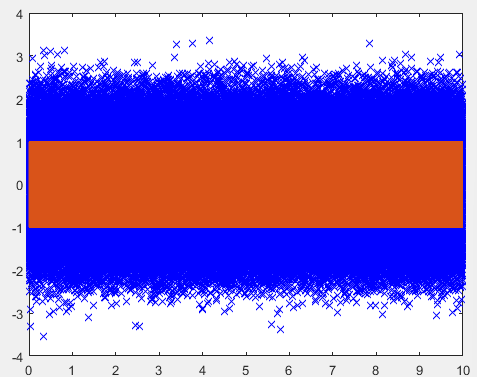


Rayleigh Fading channel은 Bits Generation, Modulation, Constellation까지 동일한 코드로 작성되었다.

이후 Channel과 Noise를 생성하는 부분에서 차이를 두었다.

Rayleigh Fading Channel의 경우 Modulation 된 Signal이 Channel을 지나가고 Noise가 더해지는 상황을 가정하므로 위와 같은 수식으로 작성되었다.

이후 Equalizing을 하여 channel에 대한 보상을 진행하였다.



-Channel만 존재하는 경우 Equalizing 전 -Equalizing 후

스크린샷, 텍스트, 라인, 다채로움이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

-Channel+Noise인 경우 Equalizing 전 -Equalizing 후

EbNo을 20으로 고정한 위 그래프에서 볼 수 있듯 Noise가 더해진 후 Equalizing하는 경우에는 Noise에 의해 파란색 x가 Bits에서 벗어나 먼 곳에 나타나는 것을 확인할 수 있다.

이는 Noise에 의해 Channel을 다시 나눠주는 연산이 channel만 보상하지 못하기 때문으로 생각된다.

-Eb/No : 0텍스트, 스크린샷, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명-Eb/No : 20

Eb/No이 작은 경우 더 많은 Bits가 Threshold를 넘어가는 것을 확인할 수 있었다.

또한 Rayleigh Fading Channel이 적용된 경우가 Noise만 더해진 경우보다 더 많은 Bits가 Threshold를 넘어가는 것을 확인할 수 있었다.

이를 통해 BER이 증가해 더 안 좋은 특성을 보일 것을 예측해 볼 수 있다.

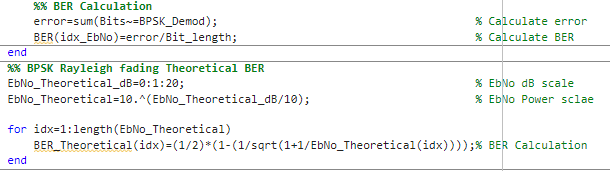
**<Demodulation>**



Equalization을 진행한 Signal을 Demodulation을 진행하였다.

Threshold인 0을 기준으로 1인경우 1로, 0인경우 그대로 0으로 저장했다.

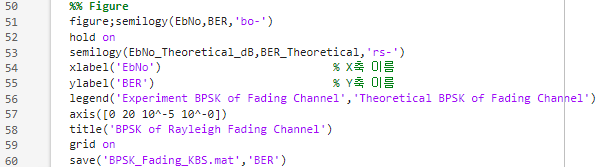
**<BER>**

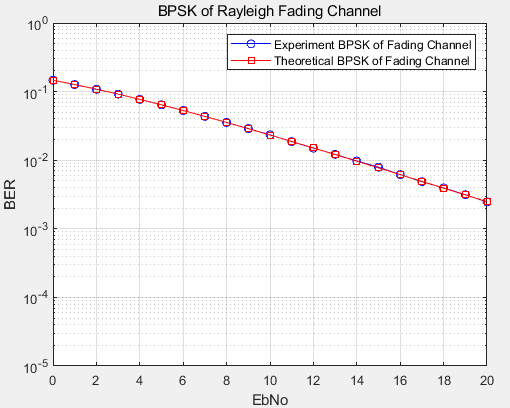


Send Bits와 Receive Bits를 비교하여 Error가 몇 개 발생하였는지 누적하고 확률로 나타내었다.

이후 이론적인 Eb/No을 생성하고 이론적인 BER을 계산하였다.

**<Figure>**

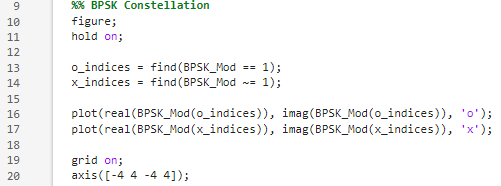


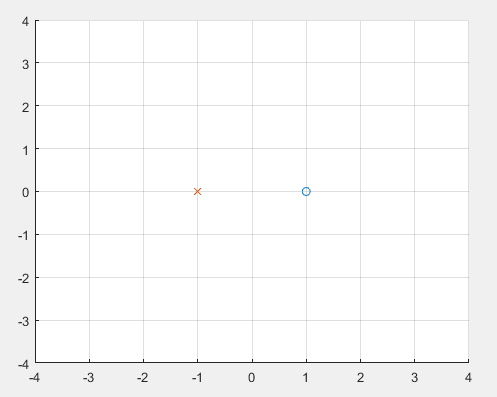


이론적인 BER vs Eb/No Curve와 동일한 것을 확인할 수 있었다.

약간의 오차는 Bits Length를 키우면 더 줄어들 것으로 생각된다.

**<Constellation 검증>**





Constellation이 제대로 나오고 있는지 확인하기 위해 find 함수를 사용하여 1인 경우가 1로 되고 0인 경우가 -1로 제대로 mapping되고 있는지 확인하였다.

두 경우 모두 제대로 나오고 있는 것을 확인할 수 있었다.

**<AWGN, Rayleigh Fading Plot>**

텍스트, 라인, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명

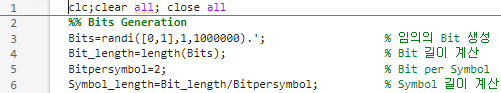
코드를 활용하여 10-5까지 BER이 낮아지는 Eb/No을 구해보았다.

AWGN만 존재하는 경우 약 9dB로 나타났으며 Rayleigh Fading Channel이 추가된 경우 약 43dB가 필요한 것으로 나타났다.

즉 Rayleigh Fading Channel이 추가된 경우 매우 큰 에너지가 필요한 것을 알 수 있다.

**2-3. QPSK Simulation (AWGN)**

**<Bits Generation>**



먼저 송수신에 사용할 Bits를 0과 1로 구성된 Random한 정수로 107개 생성해 주었다.

이는 Target BER인 10-5을 맞추기 위함이다.

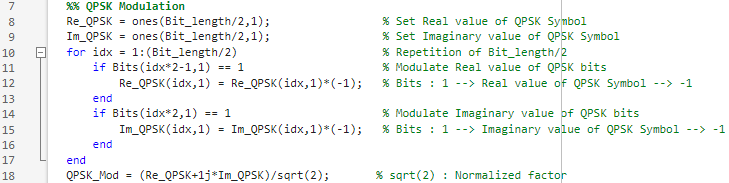
100개 이상의 표본이 쌓이면 신뢰할 수 있다고 판단할 수 있기 때문에 107로 설정하였다.

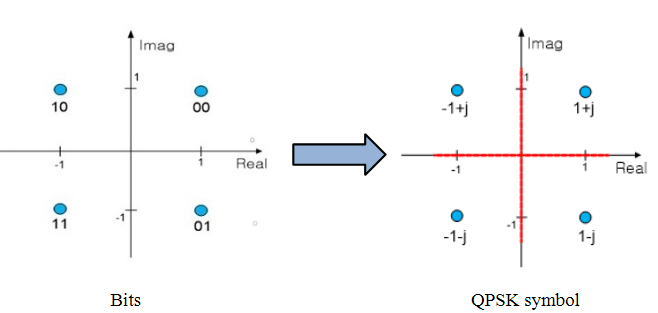
이후 Length()함수를 이용해서 생성된 Bits의 길이를 구하였다.

BPSK는 하나의 Symbol이 2개의 Bit를 표현하므로, Bit per Symbol은 2로 선언하였다.

Symbol의 길이는 Bits를 Bit per Symbol로 나누면 되므로 이를 수식으로 작성하여 Symbol의 길이를 구하였다.

**<Modulation>**





Modulation은 위 코드를 통하여 구현하였다.

홀수 번째 Bit가 1이면 Real 축에 -1을 곱하고, 짝수 번째 Bit가 1이면 Imag축에 -1을 곱해주었다.

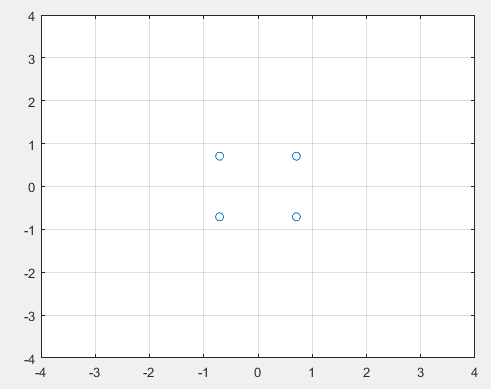
이를 통해 홀수 번째 Bit로 Y축을 기준으로 구별하고 짝수 번째 Bit로 X축을 기준으로 구별하여 값을 설정해주었다.

이 때 Modulation 값은 Normalized Factor인 root(2)를 나눠주어 진행하였다.

이를 통해 송신 파워를 1로 Normalize 하였다.

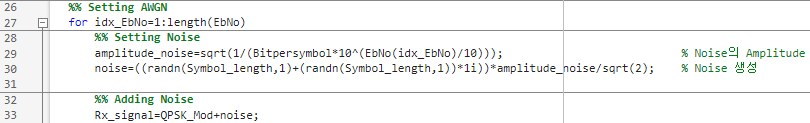
**<Constellation>**



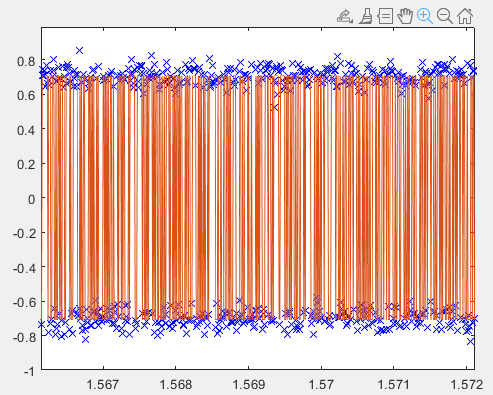
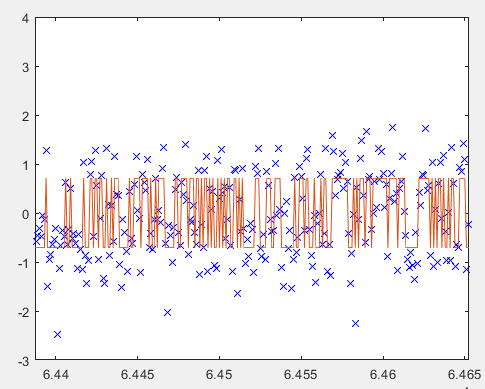


위 코드를 통해 만든 Constellation을 Plot하였다.

**<Setting AWGN>**



위 코드는 EbNo을 0부터 20까지 1씩 증가하도록 생성 후 EbNo에 따라 다른 Noise를 생성하도록 설정하였다.

-Eb/No : 0  -Eb/No : 20

위 Eb/No에 따른 그래프에서 볼 수 있듯 Eb/No이 0인 경우 빨간색 Modulation된 Bits에서 벗어난 파란색 x를 볼 수 있다.

이는 Noise의 Amplitude가 커지면서 Noise가 더해진 Signal이 큰 편차를 보이는 것으로 생각해볼 수 있다.

반대로 Eb/No이 20인 경우 Bits 값과 유사한 곳에 파란색 x 표시가 존재하는 것을 확인해 볼 수 있다.

이는 Noise의 Amplitude가 작아지면서 Noise가 더해진 Signal이 작은 편차를 보이는 것으로 생각해 볼 수 있다.

-Eb/No : 0 도표, 텍스트, 스크린샷, 원이(가) 표시된 사진

자동 생성된 설명도표, 텍스트, 그래프, 스크린샷이(가) 표시된 사진

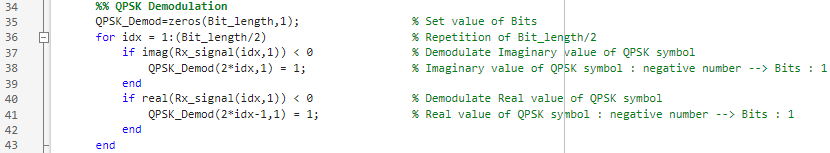
자동 생성된 설명-Eb/No : 20

위 결과는 Constellation에서 살펴보면 위와 같은 결과를 얻을 수 있다.

Decision Boundary를 살펴보면 Eb/No이 작은 쪽이 더 많은 영역을 침범하는 것을 확인할 수 있다.

따라서 Eb/No이 작은 경우 BER이 높을 것을 생각해볼 수 있다.

**<Demodulation>**



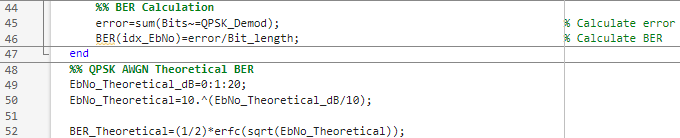
BPSK에서 Threshold는 X축과 Y축을 기준으로 한다.

이는 각 Constellation상의 Symbol에서의 최소 거리의 절반을 기준으로 하기 때문이다.

이 기준을 통해 Real+Imag\*j로 이루어진 값을 Bits로 다시 복원하였다.

Demodulation 된 값을 저장하기 위하여 Bit length 만큼의 열을 가지는 행렬을 0으로 초기화하여 생성하고 0을 기준으로 0보다 크면 1을 대입하고 0보다 크지 않은 경우 0으로 그대로 저장된다.

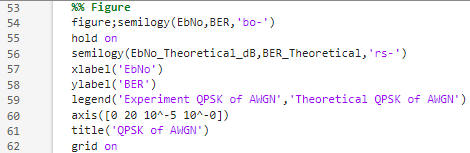
**<BER>**



BER은 Receive Bits와 Send Bits를 비교하고 다른 개수를 더해서 저장한 후 Bits의 length로 나눠 비율을 구하였다.

이후 이론적인 값과 비교하기 위하여 Eb/No을 설정하고 수식에 대입하여 값을 저장해주었다.

**<Figure>**



위 코드를 통하여 코드로 작성한 부분과 이론적인 부분을 비교하여 Plot하였다.

텍스트, 그래프, 라인, 도표이(가) 표시된 사진

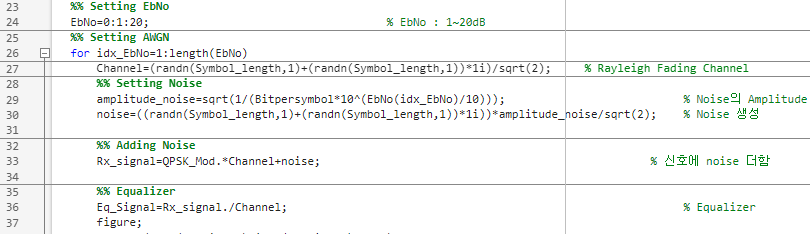
자동 생성된 설명

이론적인 내용과 매우 유사하게 나타난 것을 확인할 수 있었다.

약간의 차이를 보이는 부분은 Bits의 길이를 더 길게 하면 줄어들 것으로 생각된다.

**2-4. BPSK Simulation (Rayleigh Fading)**

**<Setting Channel & Noise>**

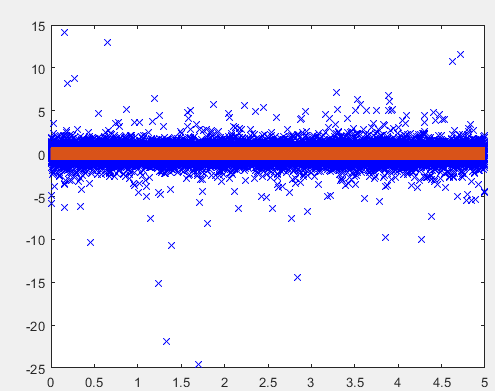
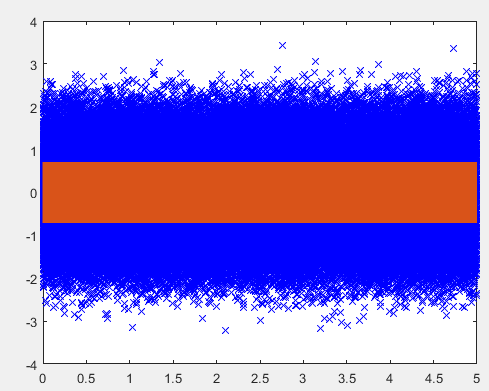


Rayleigh Fading channel은 Bits Generation, Modulation, Constellation까지 동일한 코드로 작성되었다.

이후 Channel과 Noise를 생성하는 부분에서 차이를 두었다.

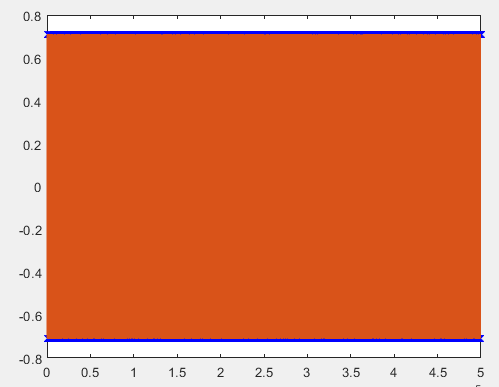
Rayleigh Fading Channel의 경우 Modulation 된 Signal이 Channel을 지나가고 Noise가 더해지는 상황을 가정하므로 위와 같은 수식으로 작성되었다.

이후 Equalizing을 하여 channel에 대한 보상을 진행하였다.



-Channel만 존재하는 경우 Equalizing 전 -Equalizing 후

텍스트, 스크린샷, 라인, 다채로움이(가) 표시된 사진

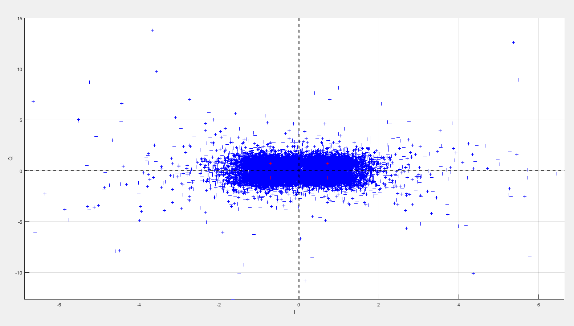
자동 생성된 설명

-Channel+Noise인 경우 Equalizing 전 -Equalizing 후

벗어나 먼 곳에 나타나는 것을 확인할 수 있다.

이는 Noise에 의해 Channel을 다시 나눠주는 연산이 channel만 보상하지 못하기 때문으로 생각된다.

-Eb/No : 0 텍스트, 스크린샷, 라인, 다채로움이(가) 표시된 사진

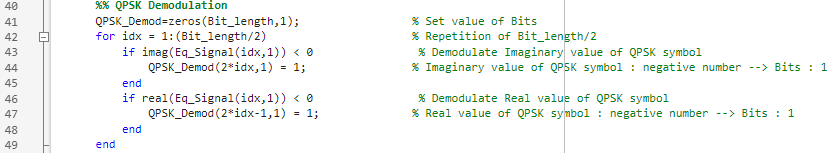
자동 생성된 설명 -Eb/No : 20

Eb/No이 작은 경우 더 많은 Bits가 Threshold를 넘어가는 것을 확인할 수 있었다.

또한 Rayleigh Fading Channel이 적용된 경우가 Noise만 더해진 경우보다 더 많은 Bits가 Threshold를 넘어가는 것을 확인할 수 있었다.

이를 통해 BER이 증가해 더 안 좋은 특성을 보일 것을 예측해 볼 수 있다.

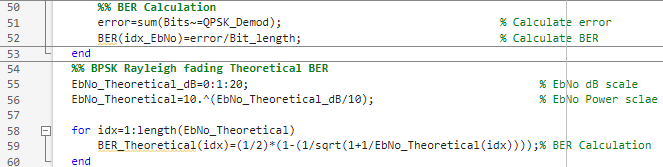
**<Demodulation>**



Equalization을 진행한 Signal을 Demodulation을 진행하였다.

Threshold인 X축과 Y축을 기준으로 하였다.

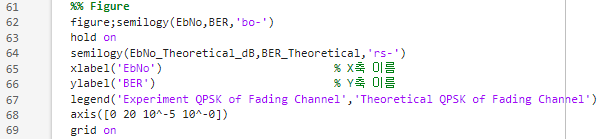
**<BER>**

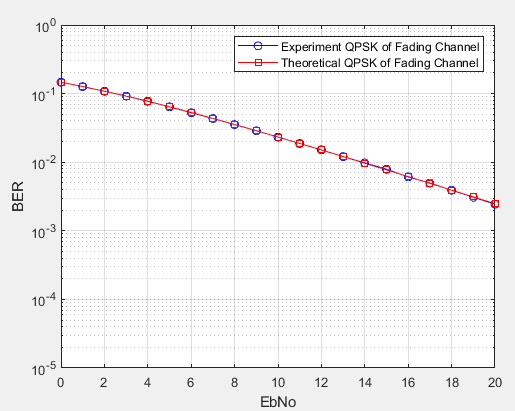


Send Bits와 Receive Bits를 비교하여 Error가 몇 개 발생하였는지 누적하고 확률로 나타내었다.

이후 이론적인 Eb/No을 생성하고 이론적인 BER을 계산하였다.

**<Figure>**





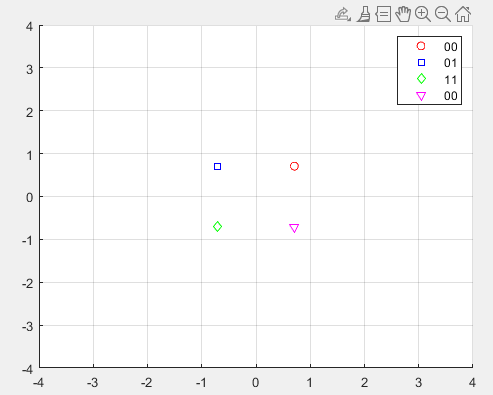
이론적인 BER vs Eb/No Curve와 동일한 것을 확인할 수 있었다.

약간의 오차는 Bits Length를 키우면 더 줄어들 것으로 생각된다.

**<Constellation 검증>**

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명



Constellation이 제대로 나오고 있는지 확인하였다.

X축과 Y축을 경계로 설정했던 값이 모두 제대로 Mapping되고 있는 것을 확인할 수 있었다.

**<AWGN, Rayleigh Fading Plot>**

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

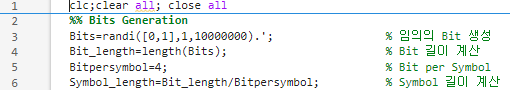
코드를 활용하여 10-5까지 BER이 낮아지는 Eb/No을 구해보았다.

AWGN만 존재하는 경우 약 9dB로 나타났으며 Rayleigh Fading Channel이 추가된 경우 약 43dB가 필요한 것으로 나타났다.

즉 Rayleigh Fading Channel이 추가된 경우 매우 큰 에너지가 필요한 것을 알 수 있다.

**2-5. 16QAM Simulation (AWGN)**

**<Bits Generation>**



먼저 송수신에 사용할 Bits를 0과 1로 구성된 Random한 정수로 107개 생성해 주었다.

이는 Target BER인 10-5을 맞추기 위함이다.

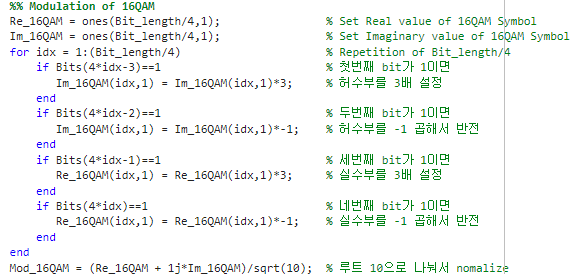
100개 이상의 표본이 쌓이면 신뢰할 수 있다고 판단할 수 있기 때문에 107개로 설정하였다.

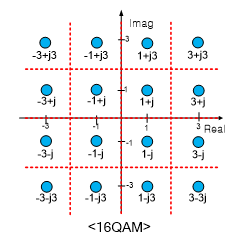
이후 Length()함수를 이용해서 생성된 Bits의 길이를 구하였다.

BPSK는 하나의 Symbol이 4개의 Bit를 표현하므로, Bit per Symbol은 4로 선언하였다.

Symbol의 길이는 Bits를 Bit per Symbol로 나누면 되므로 이를 수식으로 작성하여 Symbol의 길이를 구하였다.

**<Modulation>**



텍스트, 폰트, 번호, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Modulation은 위 코드를 통하여 구현하였다.

오른쪽 그림과 같이 Symbol을 구별하여 Mapping 하였다.

이 때 Modulation은 root(10)을 나눠 진행해주었다.

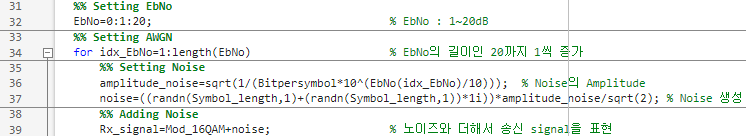
이는 Power를 1로 Normalize하기 위한 값이다.

**<Constellation>**

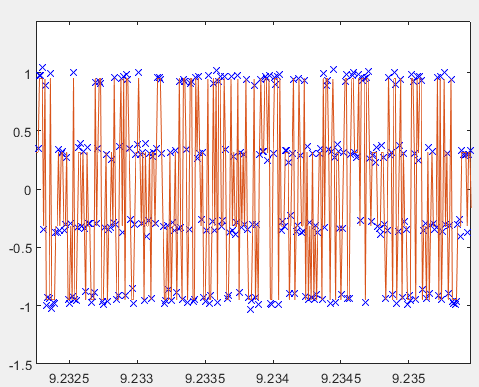
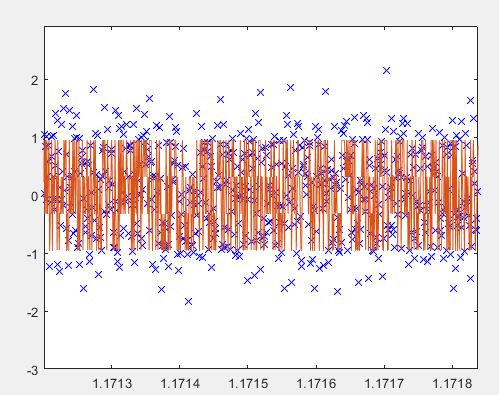


위 코드를 통해 만든 Constellation을 Plot하였다.

**<Setting AWGN>**



위 코드는 EbNo을 0부터 20까지 1씩 증가하도록 생성 후 EbNo에 따라 다른 Noise를 생성하도록 설정하였다.

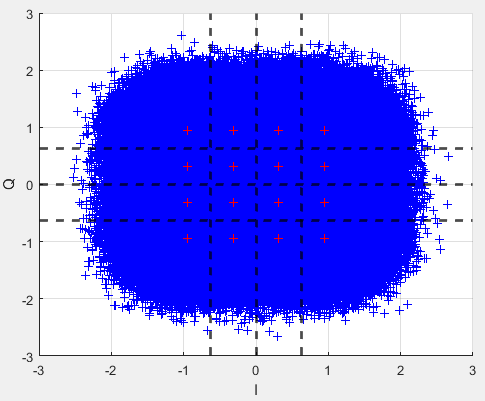
Eb/No : 0- -Eb/No : 20

위 Eb/No에 따른 그래프에서 볼 수 있듯 Eb/No이 0인 경우 빨간색 Modulation된 Bits에서 벗어난 파란색 x를 볼 수 있다.

이는 Noise의 Amplitude가 커지면서 Noise가 더해진 Signal이 큰 편차를 보이는 것으로 생각해볼 수 있다.

반대로 Eb/No이 20인 경우 Bits 값과 유사한 곳에 파란색 x 표시가 존재하는 것을 확인해 볼 수 있다.

이는 Noise의 Amplitude가 작아지면서 Noise가 더해진 Signal이 작은 편차를 보이는 것으로 생각해 볼 수 있다.

-Eb/No : 0 도표, 다채로움, 텍스트, 스크린샷이(가) 표시된 사진

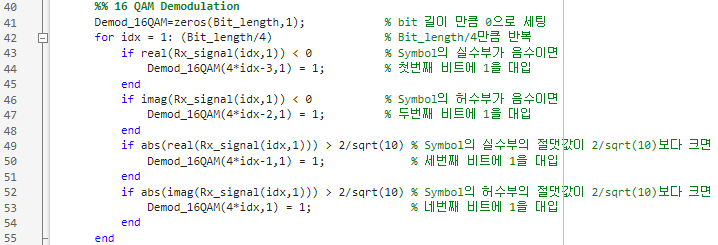
자동 생성된 설명-Eb/No : 20

위 결과는 Constellation에서 살펴보면 위와 같은 결과를 얻을 수 있다.

Decision Boundary를 살펴보면 Eb/No이 작은 쪽이 더 많은 영역을 침범하는 것을 확인할 수 있다.

따라서 Eb/No이 작은 경우 BER이 높을 것을 생각해볼 수 있다.

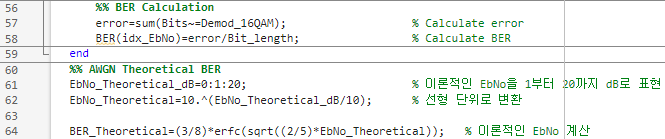
**<Demodulation>**



16QAM에서는 각 Symbol을 Gray Code Mapping이 가능하도록 구별하였다.

Demodulation 된 값을 저장하기 위하여 Bit length 만큼의 열을 가지는 행렬을 0으로 초기화하여 생성하고 각 조건에 맞춰 1을 대입하고 조건에 부합하지 않는 경우 0을 대입하였다.

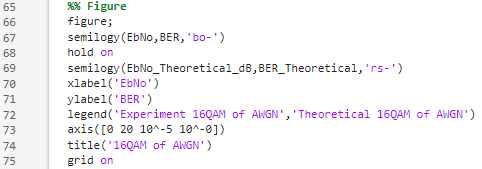
**<BER>**



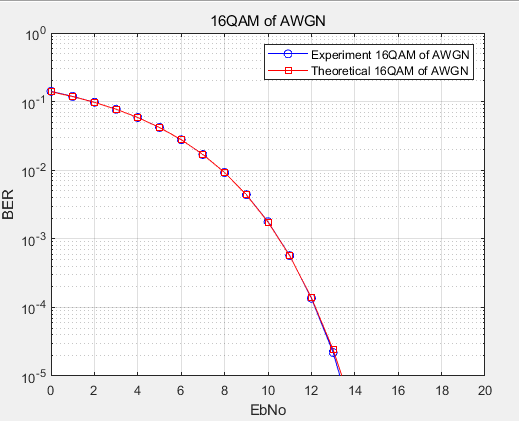
BER은 Receive Bits와 Send Bits를 비교하고 다른 개수를 더해서 저장한 후 Bits의 length로 나눠 비율을 구하였다.

이후 이론적인 값과 비교하기 위하여 Eb/No을 설정하고 수식에 대입하여 값을 저장해주었다.

**<Figure>**



위 코드를 통하여 코드로 작성한 부분과 이론적인 부분을 비교하여 Plot하였다.

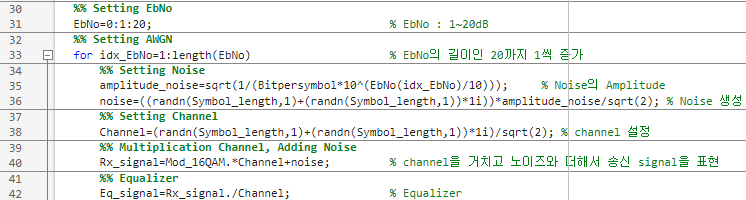


이론적인 내용과 매우 유사하게 나타난 것을 확인할 수 있었다.

약간의 차이를 보이는 부분은 Bits의 길이를 더 길게 하면 줄어들 것으로 생각된다.

**2-6. 16QAM Simulation (Rayleigh Fading)**

**<Setting Channel & Noise>**

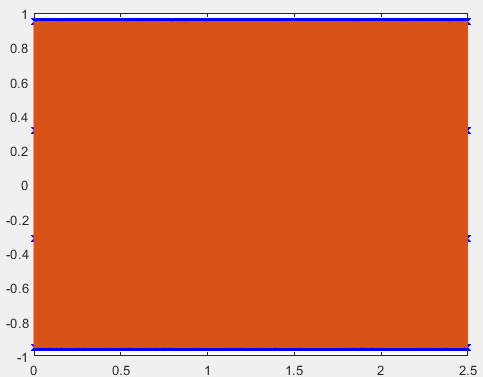
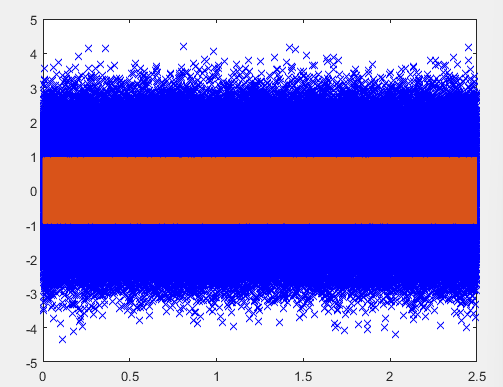


Rayleigh Fading channel은 Bits Generation, Modulation, Constellation까지 동일한 코드로 작성되었다.

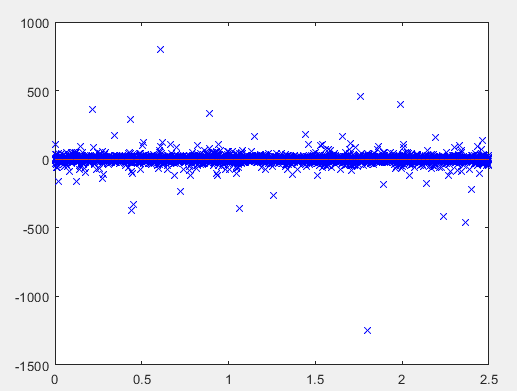
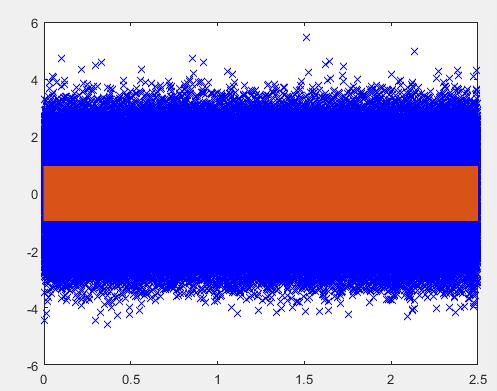
이후 Channel과 Noise를 생성하는 부분에서 차이를 두었다.

Rayleigh Fading Channel의 경우 Modulation 된 Signal이 Channel을 지나가고 Noise가 더해지는 상황을 가정하므로 위와 같은 수식으로 작성되었다.

이후 Equalizing을 하여 channel에 대한 보상을 진행하였다.



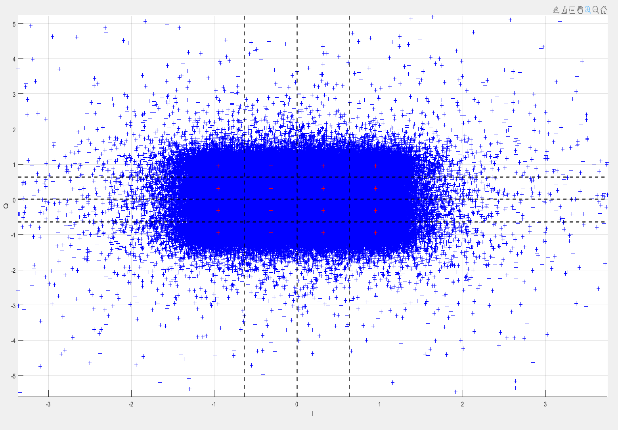
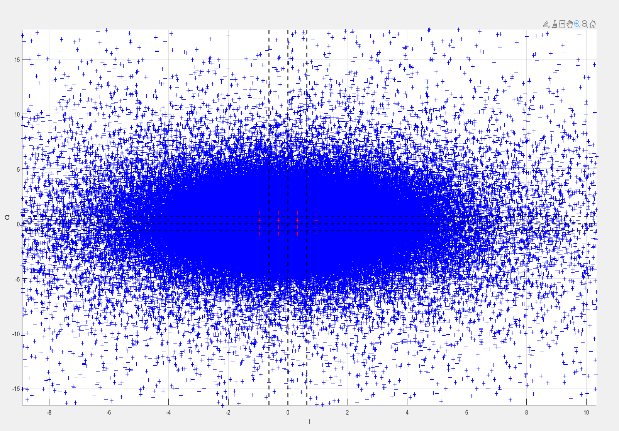
-Channel만 존재하는 경우 Equalizing 전 -Equalizing 후



-Channel+Noise인 경우 Equalizing 전 -Equalizing 후

EbNo을 20으로 고정한 위 그래프에서 볼 수 있듯 Noise가 더해진 후 Equalizing하는 경우에는 Noise에 의해 파란색 x가 Bits에서 벗어나 먼 곳에 나타나는 것을 확인할 수 있다.

이는 Noise에 의해 Channel을 다시 나눠주는 연산이 channel만 보상하지 못하기 때문으로 생각된다.

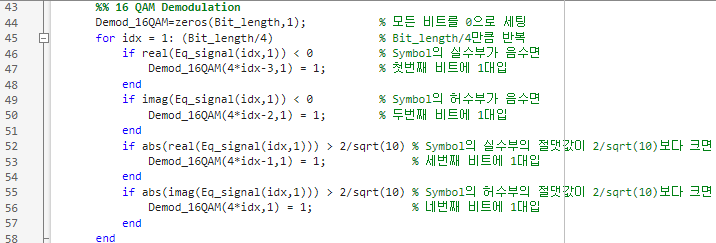
Eb/No : 0 - -Eb/No : 20

Eb/No이 작은 경우 더 많은 Bits가 Threshold를 넘어가는 것을 확인할 수 있었다.

또한 Rayleigh Fading Channel이 적용된 경우가 Noise만 더해진 경우보다 더 많은 Bits가 Threshold를 넘어가는 것을 확인할 수 있었다.

이를 통해 BER이 증가해 더 안 좋은 특성을 보일 것을 예측해 볼 수 있다.

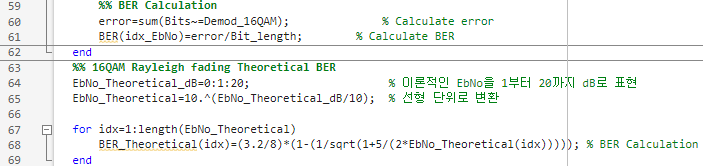
**<Demodulation>**

****

Equalization을 진행한 Signal을 Demodulation을 진행하였다.

Bits을 구별해 Real+Imag\*j로 설정한 것을 역으로 다시 Bits로 변환해 저장해주었다.

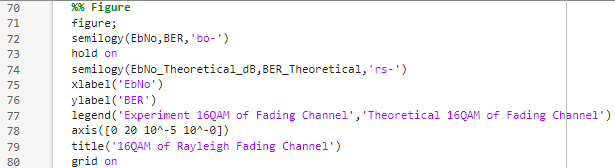
**<BER>**



Send Bits와 Receive Bits를 비교하여 Error가 몇 개 발생하였는지 누적하고 확률로 나타내었다.

이후 이론적인 Eb/No을 생성하고 이론적인 BER을 계산하였다.

**<Figure>**



이론적인 BER vs Eb/No Curve와 동일한 것을 확인할 수 있었다.

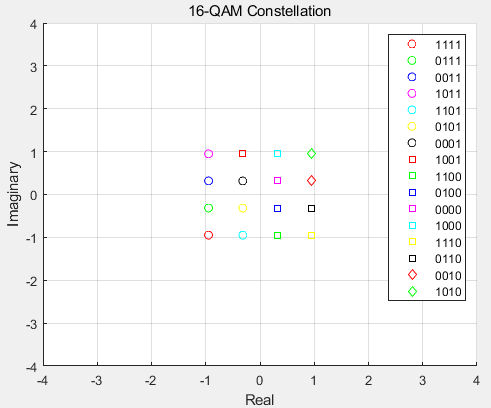
텍스트, 그래프, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

약간의 오차는 Bits Length를 키우면 더 줄어들 것으로 생각된다.

**<Constellation 검증>**





Constellation이 제대로 mapping되고 있는지 확인하였다.

모든 경우에서 제대로 Mapping되고 있는 것을 확인할 수 있었다.

**<AWGN, Rayleigh Fading Plot>**

텍스트, 라인, 그래프, 번호이(가) 표시된 사진

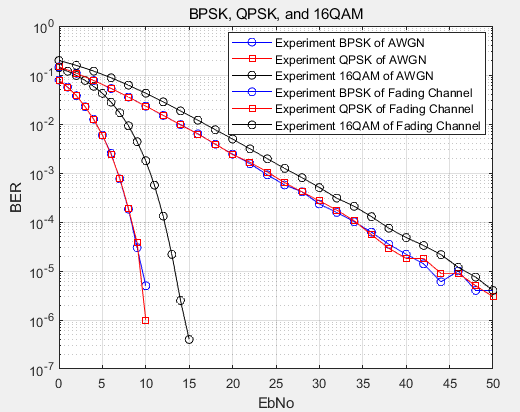
자동 생성된 설명

코드를 활용하여 10-5까지 BER이 낮아지는 Eb/No을 구해보았다.

AWGN만 존재하는 경우 약 13dB로 나타났으며 Rayleigh Fading Channel이 추가된 경우 약 46dB가 필요한 것으로 나타났다.

즉 Rayleigh Fading Channel이 추가된 경우 매우 큰 에너지가 필요한 것을 알 수 있다.

**2-7. BPSK ,QPSK, 16QAM Plot**



위 그래프를 하나의 Plot으로 나타내보았다.

BPSK와 QPSK는 같은 BER vs Eb/No을 가지므로 동일한 그래프 개형을 보이는 것을 확인할 수 있었다.

반면 16QAM은 같은 BER을 가지기 위해서는 AWGN에서 약 3dB 더 필요한 것을 확인할 수 있었다.

Rayleigh Fading Channel에서는 이렇게 큰 차이가 발생하지는 않았지만 16QAM이 더 큰 Eb/No이 필요한 것을 확인해볼 수 있었다.

이는 아래 식을 통해 원인을 분석할 수 있다.

텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명

위 식에서 볼 수 있듯 BPSK와 QPSK는 같은 Eb/No을 가진다.

따라서 같은 Curve를 보이게 된다.

16QAM의 경우에는 Data Rate이 높지만 Minimum Distance가 감소하기 때문에 BER이 증가하게 되어 BER 성능이 열화되는 것으로 생각된다.

Target BER(10-5)에서의 Eb/No을 정리하면 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| **Eb/No, 단위 : dB** | **BPSK** | **QPSK** | **16QAM** |
| **AWGN** | **10** | **10** | **13** |
| **Rayleigh Fading** | **44** | **44** | **47** |

1. **BPSK, QPSK, 16QAM with Linear Block Code (7,4)**

Linear Block Code는 Channel Encoding에 적용되며 Modulation 전에 적용되어야 하고 Decoding 과정은 Demodulation 후에 진행되어야 한다.

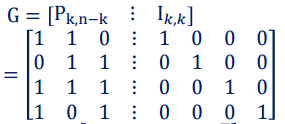
따라서 Random Bit를 생성하고 Codeword Generation을 진행하였다.

(7,4)의 Hamming Code를 사용하여 4Bits의 Message 정보를 담고 있도록 Vector를 생성하고 Generator Matrix : G와 곱하여 Codeword를 생성하였다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

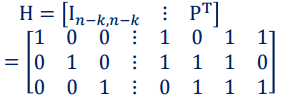
자동 생성된 설명

-Message Word와 CodeWord 변환



-Generator Matrix

생성된 Codeword들은 Channel을 지나고 noise가 더해진 다음 Equalizer를 통해 r로 변환된다.



-Parity Check Matrix

이때 수신된 r은 error를 포함하고 있기 때문에 HT를 곱하여 Syndrome을 구한다.

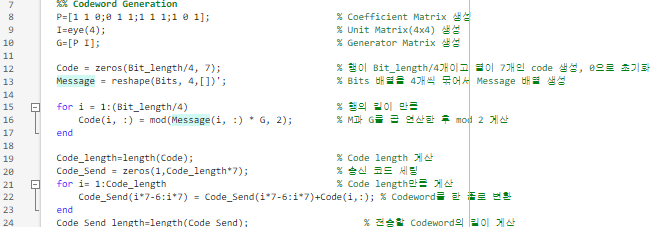
이때 구한 Syndrome을 통해 Error Pattern을 구하고 r에 더해주어 원래 송신된 Bits로 표현한다.

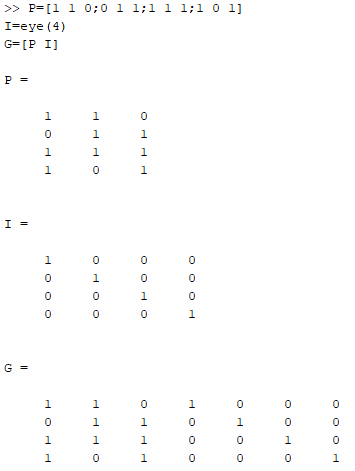
텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

-Syndrome과 Error Pattern -Syndrome -received Signal

위 내용과 앞서 작성한 코드를 기반으로 Linear Block Code를 적용하였다.





먼저 Coefficient Matrix와 4x4 단위 행렬을 선언하였다.

이 두 행렬을 결합하여 Generator Matrix를 생성하였다.

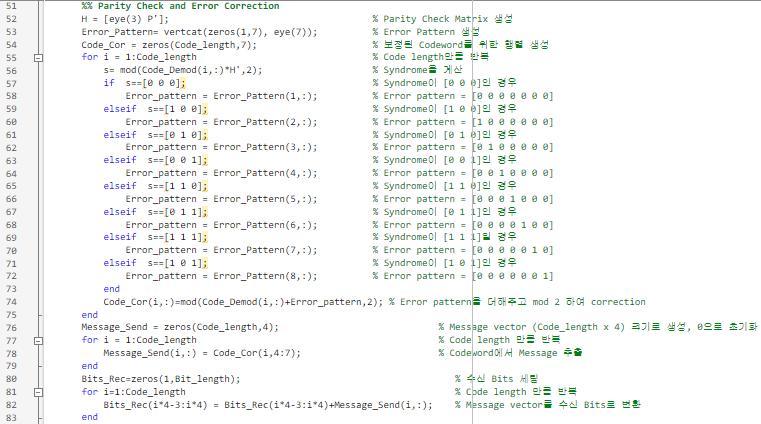
Code는 7의 길이를 가지고 Message는 Bits를 4개의 행으로 잘라서 Transpose하는 형식으로 열이 4이고 행이 Bits/4이 되도록 생성하였다.

Codeword는 Message와 G의 곱을 Mod 2한 것과 같으므로 모든 행에 대해 위 연산을 진행하였다.

Codeword를 한 줄로 변환하여 Modulation을 진행하고 전송하는 상황을 가정하기 위해 한 줄로 변환하는 코드를 작성하였다.

위 코드를 통해 생성된 Codeword를 BPSK, QPSK, 16QAM으로 각각 Modulation한 후 AWGN과 Rayleigh Fading Channel을 적용하고 Equalizer를 수행한 후 Demodulation을 진행하였다.

**<Parity Check & Error Correction>**



Demodulation을 마친 Codeword는 Error를 포함하고 있기 때문에 Error를 찾아내고 이를 더해 Message를 복원해내는 작업을 거쳐야 한다.

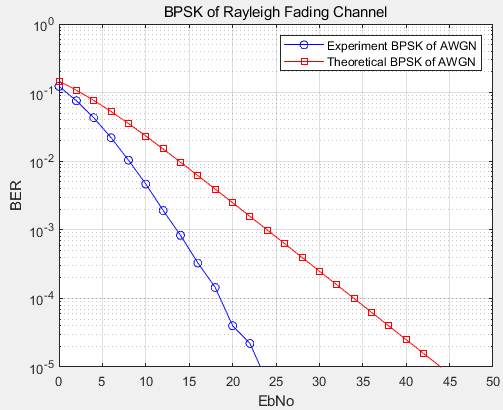
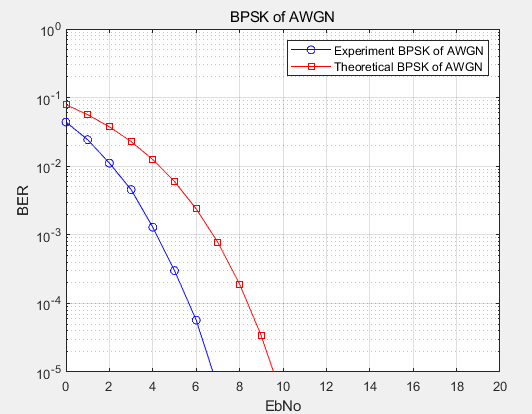
Error를 복원하기 위해 Parity Check Matrix를 생성하고 Demodulation된 Codeword에 곱해 Mod 2 연산을 하여 Syndrome을 구해주었다.

또한 주어진 Table에 의해 Error Pattern을 찾아주고 이를 Codeword에 더해주어 최종적으로 BER을 구할 Signal을 복원하였다.

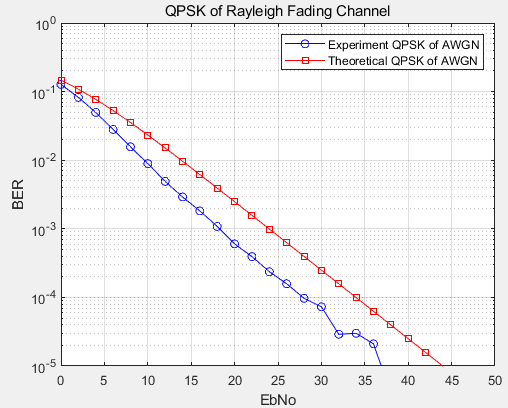
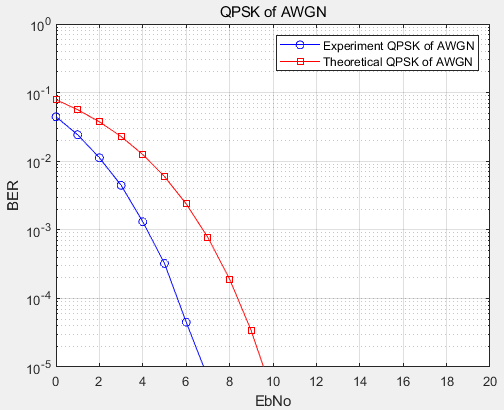
Linear Block Code는 Systematic Code이므로 Message를 Code에서 바로 추출하고 이를 하나의 Vector로 세우는 과정을 진행하였다.

**<Plot>**

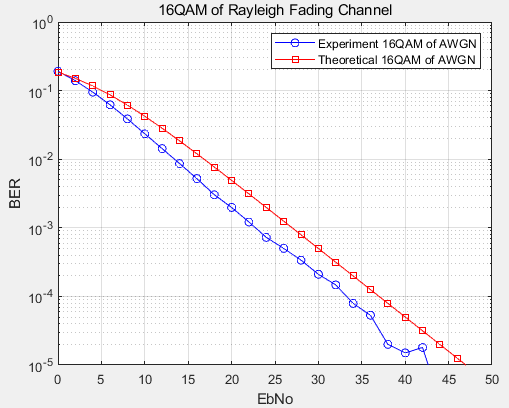
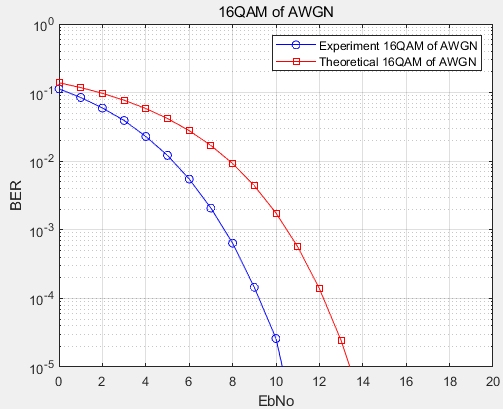
**-BPSK**

****

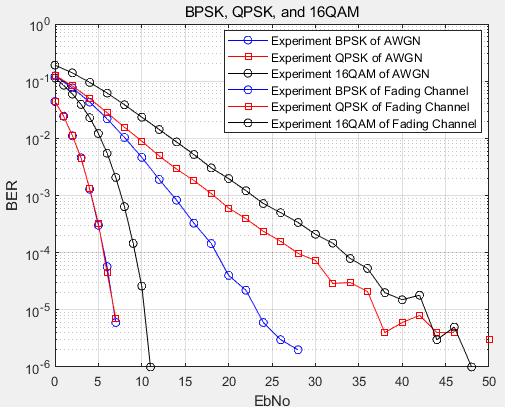
**-QPSK**

****

**-16QAM**

****

**-BPSK, QPSK, 16QAM**

****

Target BER인 10-5을 기준으로 Eb/No을 비교해보면 아래 표와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| **Eb/No, 단위 : dB** | **BPSK** | **QPSK** | **16QAM** |
| **AWGN** | **6** | **6** | **11** |
| **Rayleigh Fading** | **23** | **36** | **43** |

위 표를 통해 Linear Block Encoding을 하지 않은 경우와 비교하면 아래와 같은 Eb/No 차이를 보인다.

|  |  |  |  |
| --- | --- | --- | --- |
| **Delta Eb/No, 단위 : dB** | **BPSK** | **QPSK** | **16QAM** |
| **AWGN** | **-4** | **-4** | **-2** |
| **Rayleigh Fading** | **-21** | **-8** | **-4** |

따라서 주어진 Encoding 방식이 Power를 좀 더 효율적으로 사용할 수 있게 만든다고 할 수 있다.

1. **BPSK, QPSK, 16QAM with Convolutional Code**

Convolutional Code는 Trellis를 지나가며 부호를 생성하는 방식을 말한다.

Convolutional Code는 Matlab의 Built-in Function인 poly2trellis()과 convenc()을 사용하여 구현하였다.

도표, 스케치, 그림, 기술 도면이(가) 표시된 사진

자동 생성된 설명

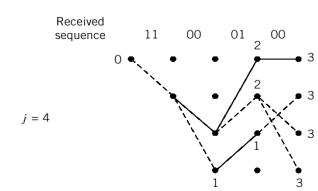
-Trellis

poly2trellis()은 Trellis Structure를 제공하는 함수이다.

Code Rate가 1/2이고 , 으로 작성하여야 하므로 이를 구현할 수 있는 Trellis Structure를 사용하였다.

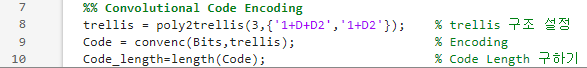
Decoding은 vitdec() 함수를 사용하였다.

이 함수는 Trellis Structure, Traceback Depth, Operating Mode, Decoding Type을 매개변수로 하는 함수로 Viterbi Algorithm을 사용하여 Decoding을 진행하였다.

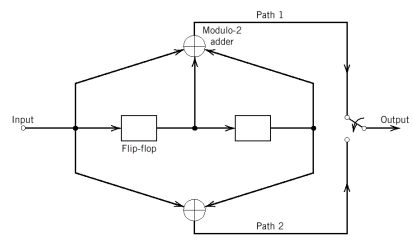


-Viterbi Algorithm

**<Encoding>**



Code Rate : r = 1/2, Polynomial : , 의 조건을 만족시키기 위하여 위와 같이 인자를 넣어 작성하였다.



**<Decoding>**



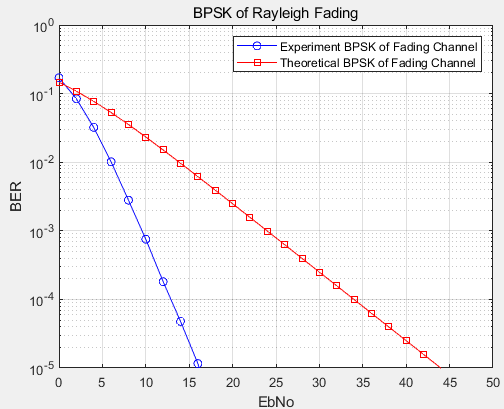
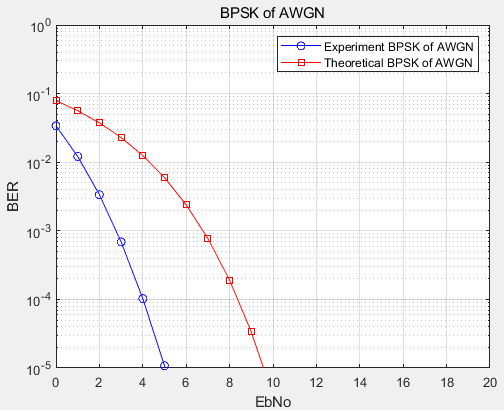
Decoding은 vitdec() 함수를 사용하였다.



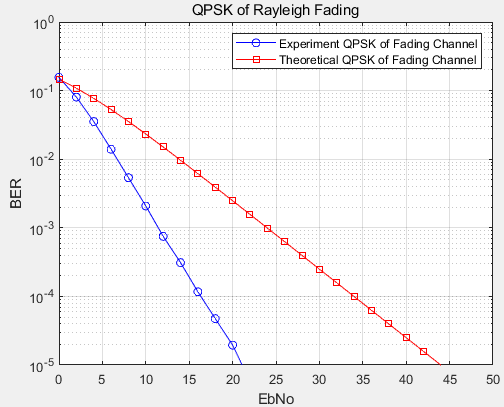
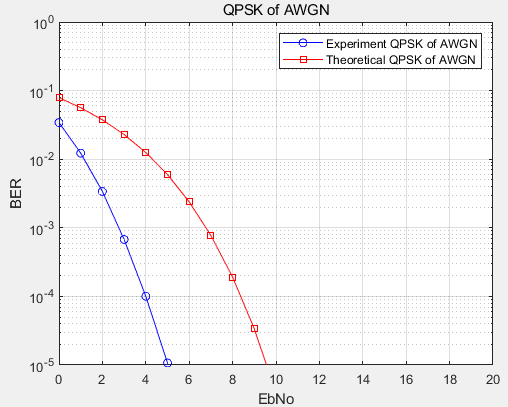
인자를 결정하는 부분은 vitdec() 함수의 설명을 참고하여 설정하였다.

**<Plot>**

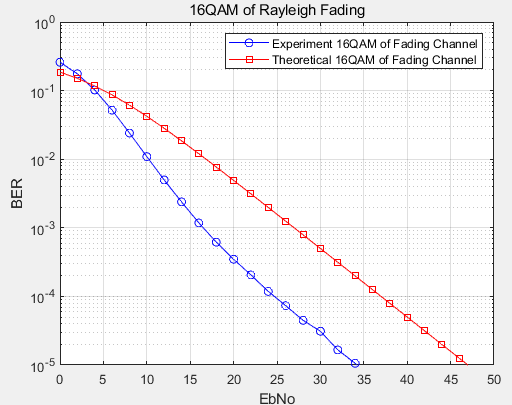
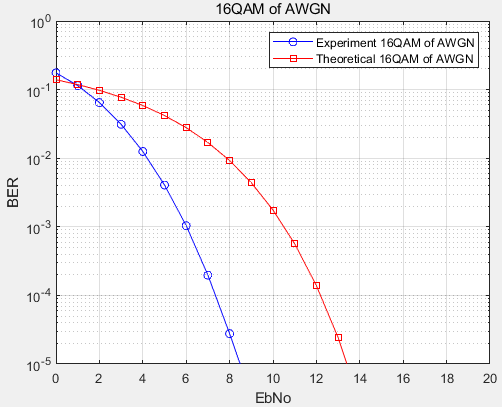
**-BPSK**



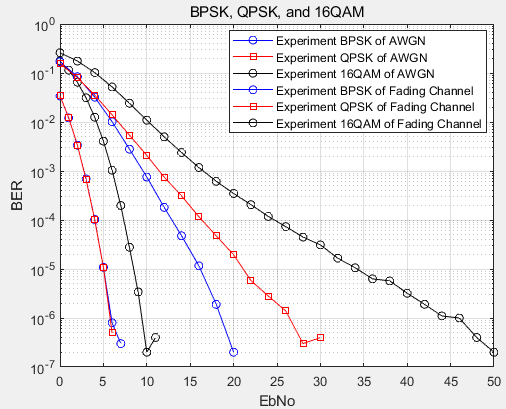
**-QPSK**



**-16QAM**



**-BPSK, QPSK, 16QAM**



Target BER인 10-5을 기준으로 Eb/No을 비교해보면 아래 표와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| **Eb/No, 단위 : dB** | **BPSK** | **QPSK** | **16QAM** |
| **AWGN** | **5** | **5** | **8** |
| **Rayleigh Fading** | **16** | **21** | **34** |

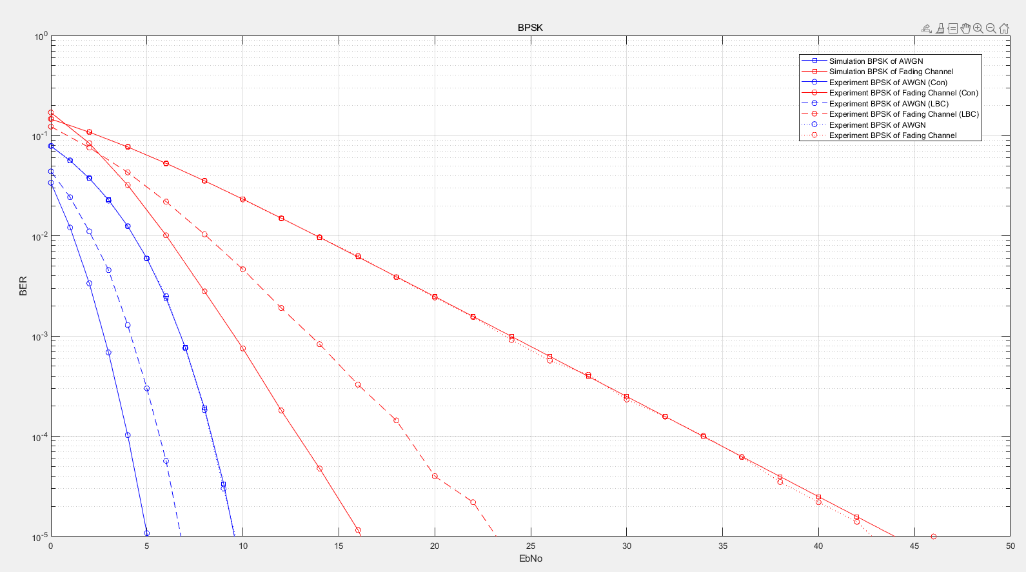
위 표를 통해 Linear Block Encoding을 한 경우와 비교하면 아래와 같은 Eb/No 차이를 보인다.

|  |  |  |  |
| --- | --- | --- | --- |
| **Delta Eb/No, 단위 : dB** | **BPSK** | **QPSK** | **16QAM** |
| **AWGN** | **-1** | **-1** | **-3** |
| **Rayleigh Fading** | **-7** | **-15** | **-9** |

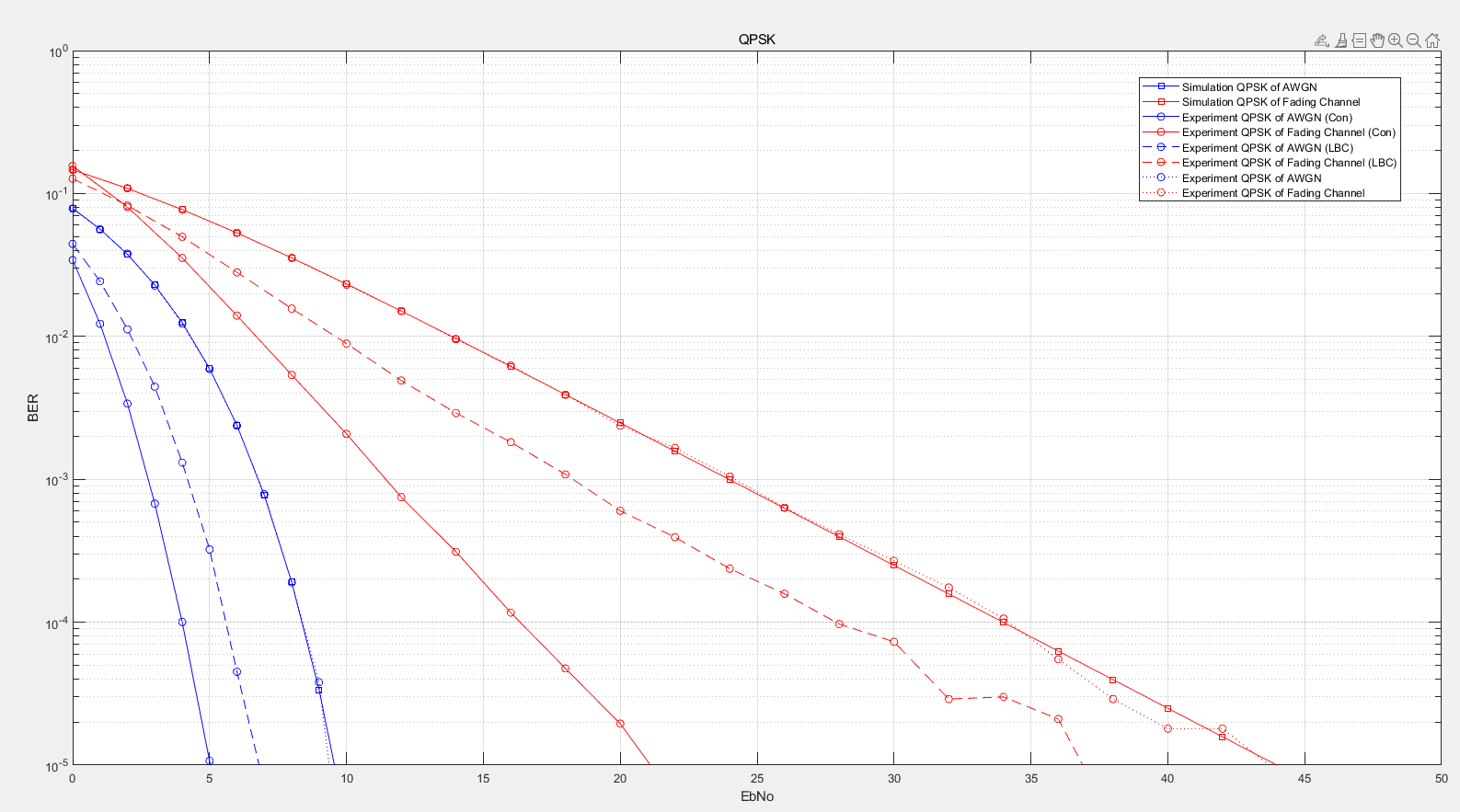
즉 Linear Block Encoding보다 더 낮은 Eb/No으로 Target BER을 달성해 Power를 적게 소모하고 신호를 전송할 수 있는 방법이라는 것을 알 수 있었다.

1. **FEC(Forward Error Correction) 방법에 따른 BER 분석**

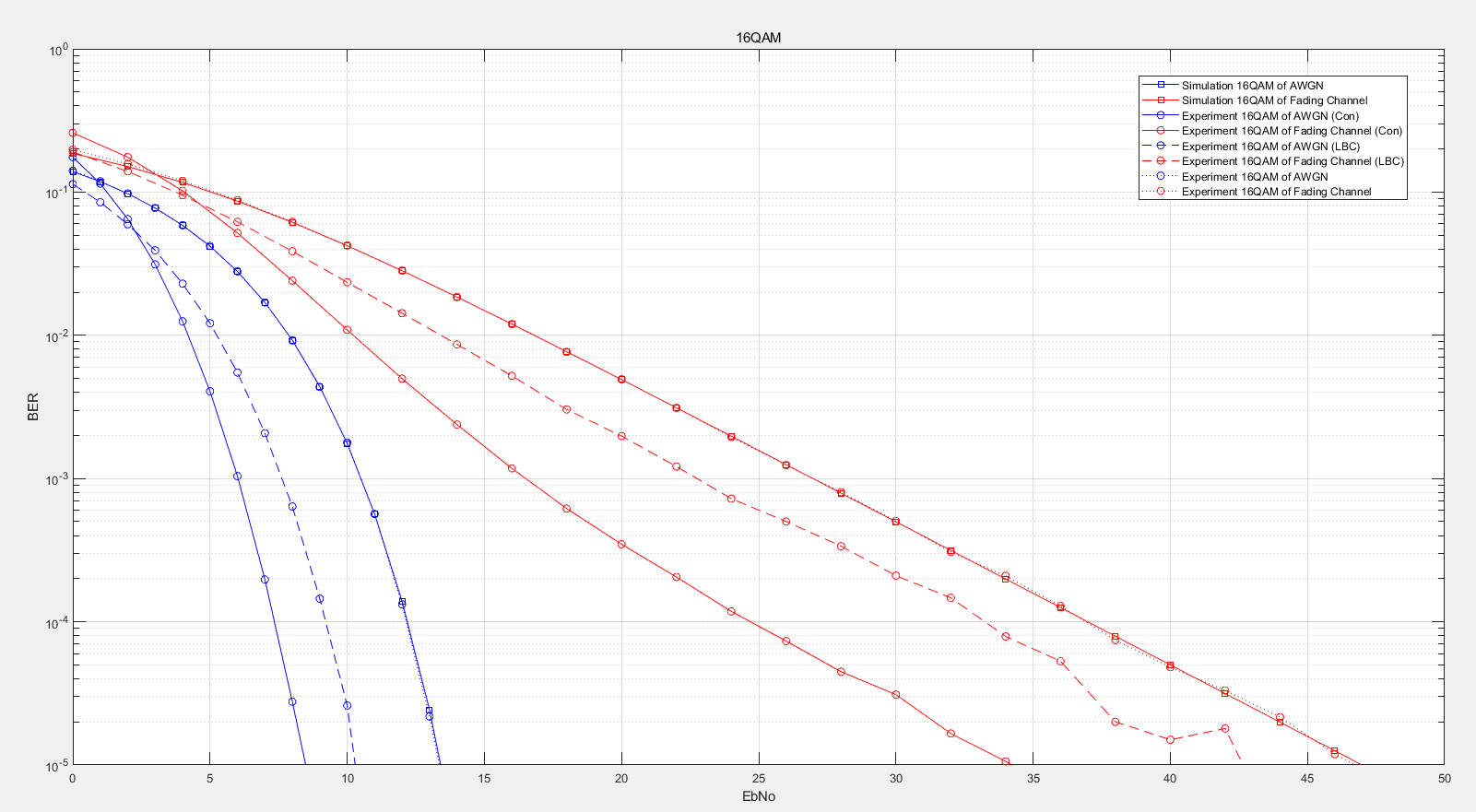
**<BPSK>**



**<QPSK>**



**<16QAM>**



Target BER이 10-5일 때 아래와 같이 정리할 수 있다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Eb/No, 단위 : dB** | **BPSK Uncoded** | **BPSK LBC** | **BPSK**  **Conv** | **QPSK Uncoded** | **QPSK LBC** | **QPSK**  **Conv** | **16QAM Uncoded** | **16QAM LBC** | **16QAM Conv** |
| **AWGN** | **10** | **6** | **5** | **10** | **6** | **5** | **13** | **11** | **8** |
| **Rayleigh Fading** | **44** | **23** | **16** | **44** | **36** | **21** | **47** | **43** | **34** |

전체적인 경향은 Uncoded<Linear Block Code<Convolutional Code 순으로 BER 성능이 좋았다.

이는 FEC로 인해 복잡도가 증가하는 대신 일정 오류를 보정할 수 있는 Capability를 얻었기 때문이다.

특히 Convolutional Code는 Memory로 이전 값과 현재 값을 연관하여 Code를 생성하고 Decoding하기 때문에 이와 같은 결과를 얻을 수 있다.

1. **고찰**

이번 Project를 통하여 FEC와 Modulation 방식에 따른 BER vs Eb/No을 비교해 볼 수 있었다.

전체적인 경향은 Uncoded<Linear Block Code<Convolutional Code 순으로 좋은 BER을 보였다.

여기까지만 보면 Convolutional Code가 가장 좋은 FEC 방식으로 보이지만 Convolutional Code는 에러가 누적된다는 단점이 있다.

Convolutional Code는 이전 값과 현재 값을 사용하여 Encoding하기 때문에 이전 값에 오류가 발생하면 계속해서 오류가 누적되게 된다.

따라서 제대로 된 Signal을 전송할 수 없게 되는 문제점이 발생할 수 있다.

Linear Block Code의 경우에는 Convolutional Code에 비해 BER 성능이 떨어진다.

하지만 Linear Block Code는 매번 연관 없는 Code를 생성하기 때문에 연속해서 오류가 발생할 수 있는 확률이 작다.

위와 같은 내용을 통해 FEC 방식을 상황에 맞게 적절하게 선택해야 한다는 것을 알 수 있었다.

다음으로 Modulation 방식에 따라 BER vs Eb/No이 크게 변화한다는 것을 알 수 있었다.

BER은 BPSK >= QPSK > 16QAM 로 좋게 나타났다.

이는 전송속도와 BER에 연관이 있다는 것을 나타낸다.

16QAM은 한 Symbol에 4Bits를 전송한다.

이에 따라 Minimum Distance도 BPSK와 QPSK에 비해 작게 나타난다.

따라서 같은 Eb/No에도 큰 BER을 갖게 되었다.

다음으로 AWGN에서는 BPSK와 QPSK가 Target BER에 대해 같은 Eb/No값을 보이지만 Rayleigh Fading Channel에서는 BPSK가 더 우월한 BER을 갖는 것을 확인할 수 있었다.

이는 BPSK는 다른 Symbol까지 하나의 거리를 갖지만 QPSK는 대각선으로 Symbol을 넘어갈 수 있기 때문에 이런 차이가 발생하는 것으로 생각된다.

대각선으로 넘어가는 경우는 2개의 Bits Error가 발생하는 경우로 생각할 수 있다.

따라서 Error rate가 높아지게 된다.

텍스트, 스크린샷, 도표, 원이(가) 표시된 사진

자동 생성된 설명텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

위 그림은 Rayleigh Fading Channel에서 Eb/No을 23dB로 설정하였을 때 Constellation과 Symbol의 값을 측정한 것이다.

이 그림을 통해 Rayleigh Fading Channel에서는 Eb/No이 바뀌어도 Symbol의 분포가 크게 바뀌지 않는다는 것을 확인할 수 있다.

따라서 대각선을 넘어간 Symbol에 의한 Error가 지속적으로 발생하게 된다.

심지어 대각선을 넘어간 Symbol을 Bit로 환산하였을 때 연속적으로 2개의 Error가 발생하게 된다.

이 때문에 Error Correction Capability가 작은 Linear Block Code에서 더 큰 BER 차이를 보이고 Convolutional Code는 Error Correction Capability가 더 크기 때문에 이보다 작은 BER 차이를 보이는 것으로 생각된다.

이 결과를 통해 상황에 맞는 Modulation 방식을 선택하여야 한다는 것을 알 수 있었다.

다음으로 Project를 진행하면서 vitdec() 함수에서 Traceback Depth 값을 설정하면서 Decoder의 복잡도가 BER 성능에 영향을 줄 수 있다는 것을 알게 되었다.

Traceback Depth는 Symbol을 Decoding하는 Path를 몇 개로 설정할 것인지 나타내는 인자이다.

따라서 많은 수를 할당하면 더 좋은 결과를 얻을 수 있다.

하지만 이렇게 되면 복잡도가 커지고 Matlab의 동작시간이 증가된다.

실제 통신 시스템을 구축할 때에도 Hardware의 복잡도가 증가하게 될 것이므로 BER과 Hardware의 복잡도 사이의 Trade off 관계를 확인할 수 있었다.

이번 Project를 통해 수업시간에 배운 이론내용을 직접 구현해보며 이해가 되지 않았던 내용들도 이해할 수 있었던 시간이었다.

1. 참고문헌

- <https://www.mathworks.com/help/comm/ref/viterbidecoder.html>

- <https://www.geeksforgeeks.org/difference-between-linear-block-codes-and-convolutional-codes/>

- <https://en.wikipedia.org/wiki/Viterbi_decoder>

-<http://www.ktword.co.kr/test/view/view.php?m_temp1=2500>

-<http://www.ktword.co.kr/test/view/view.php?m_temp1=2608&id=116>