Project #1 Report

Student ID: 101011246
Name: 李明儒

一、 模擬說明

這個報告是透過實作 Viterbi 演算法在一(2,1,6) convolutional code 上，並使其通過 AWGN 通道之表現情形。

而 information bit 滿足下列關係式：

$$u_{l+6} = u_{l+1} \oplus u_l, \qquad \text{for } l \geq 0$$

且初始情況為 $u_0 = 1, u_1 = 0,\ u_2 = 0, u_3 = 0, u_4 = 0, u_5 = 0$

接著利用(2,1,6)convolutional code 編碼，encode 成 2 個 codewords；其 generator matrix 為：

$$\boldsymbol{G}(D) = (1 + D^2 + D^3 + D^5 + D^6 \quad 1 + D + D^2 + D^3 + D^6)$$

接著把 encode 過後的 codewords 輸入 AWGN channel 裡，並加入雜訊；再加入通道雜訊後，若是選擇 hard decision，則直接將臨界點設為 0，也就是如果收到的資料是大於 0，則將其 map 為 0；反之則為 1；若為 soft decision 則直接使用接收到的資料，不須額外 mapping。接著再將這些經過 mapping 的經過 Viterbi 解碼，期間記下每個 state 上最小的 Hamming weight(若為 soft decision，則是將接收到的資料減去該時刻可能的 x1 x2 後再平方)，理論上是該跑完整個 Viterbi 演算法才一次輸出，但考慮到實作記憶體的容量有限，必須每一段時間就 truncate 一部分的 survivor，這次的報告中，truncation length 為 32，亦即每 32bit 便把最早的 bit 輸出，接著將 survivor slide 一個 bit 作為下個輸出的點，最後將這些經過解碼之後所得到的訊息，與一開始輸入的 information bit 做比較，並計算錯誤個數以及錯誤率。

二、模擬結果

Truncation length = 32

Information size = 10^5

SEED = 99

(1) hard decision

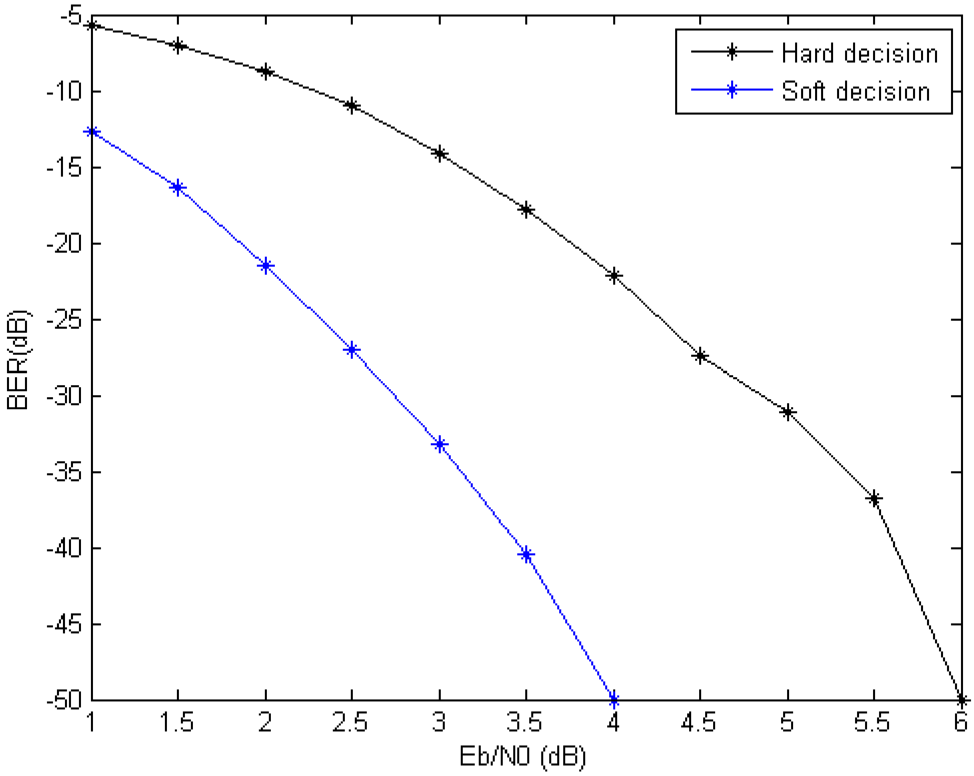
Eb/N0(dB)	1.0	1.5	2.0	2.5	3.0	3.5
Error	26501	19597	13151	7853	3852	1630
BER	2.650E-1	1.960E-1	1.315E-1	7.853E-2	3.852E-2	1.63E-2
Eb/N0(dB)	4.0	4.5	5.0	5.5	6.0	
Error	614	178	77	21	3	
BER	6.14E-3	1.78E-3	7.7E-4	2.1E-4	3E-5	

(2) soft decision

Eb/N0(dB)	1.0	1.5	2.0	2.5	3.0	3.5
Error	5371	2279	707	198	47	9
BER	5.371E-2	2.279E-2	7.07E-3	1.98E-3	4.7E-4	9E-5
Eb/N0(dB)	4.0					
Error	1					
BER	1E-5					

三、模擬結果與討論

(1)模擬結果



(2)

上圖為 BER 對 SNR 分別用 Hard decision 和 Soft decision 作圖；由上圖可以看出來，BER 隨著 SNR 上升而有明顯的下降，不論是 Hard decision 或 Soft decision。除此之外，由圖亦可看出：在同樣的 BER 下，soft decision 所需的 SNR 較 hard decision 少了 2dB 左右，會有此現象的原因是因為在 hard decision 的情況下，接收到 bit 後會先把它 map 成 0 或 1，此一動作有可能增加了此 bit 跟 information bit 的距離，進而使錯誤率提升；而相較於 hard decision，soft decision 並不會把接收到的 bit 做任何 mapping 的動作，反而是直接把他跟 trellis 上有可能的 information bit 相減後平方，這樣較不會產生太大的距離差距，所以 soft decision 在 BER 的表現上較 hard decision 出色；但 hard decision 的好處是接收器設計較為簡單，僅需判斷大於 threshold 或小於即可，故兩者皆有各自適用的地方。

四、程式碼

```
/**
 *   COM5140 Error-Correcting Codes
 *   Project1 Implementation of Viterbi Decoding
 *   ID    : 101011246
 *   NAME: Ming-Ju, Li
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>

#define N 10032 //size of the codeword

unsigned long long SEED = 5432;
unsigned long long RANV;
int RANI = 0;

void generator(unsigned long long cw1[], unsigned long long cw2[]); //the generator function
void decoder(unsigned long long int output[], unsigned long long int Y1[], unsigned long long int Y2[], 
double sigma, int mode); // the decoder function
double BER_count(unsigned long long int output[]); // find the number of the erroneous bits
int idx(int table[], int x); // find the position of a certain state
double Ranq1(void); //random number generator
double normal(double *n1, double *n2, double sigma); // normal random variables for noise

int main(void){
	double sigma, SNR;
	int i, j = 0;
	unsigned long long int X1[N/64+1], X2[N/64+1]; // the channel input
	unsigned long long int cw1[N/64+1], cw2[N/64+1]; //the encoded codeword
	unsigned long long int Y1[N/64+1] = {0} , Y2[N/64+1] = {0}; // the channel output
	int mode = -1;
	double BER;
	unsigned long long int *output1;

	output1 = (unsigned long long int*)malloc(N * sizeof(unsigned long long int));

	printf("information size: %d\n", N);
	printf("SNR: ");
```

```
	scanf("%lf", &SNR);
	printf("Decide the decision criteria(1: hard, 2: soft): ");
	scanf("%d", &mode);

	sigma = pow(10, SNR/10);
	sigma = sqrt(1/(sigma));
	printf("\nSNR is %.1lf\n",SNR);
	printf("sigma is %f\n", sigma);
	printf("\n\n\n");

	generator(cw1, cw2);

	for(i = 0; i < N / 64 + 1; i++){
		X1[i] = cw1[i];
		X2[i] = cw2[i];
	}

	decoder(output1, X1, X2, sigma, mode);

	BER = BER_count(output1) / (N - 32);
	printf("error bits: %.0f \n", BER_count(output1));
	printf("BER=%.5e", BER);

	return 0;
}

void generator(unsigned long long int cw1[], unsigned long long int cw2[]){
	int i, j = 0;
	int u[6] = {1, 0, 0, 0, 0, 0};
	int state[6] = {0};
	int x1 = 0, x2 = 0;
	unsigned long long bit[N/64+1] = {0};
	double BER;

	for(i = 0; i <= N / 64 + 1; i++){
		cw1[i] = 0;
		cw2[i] = 0;
	}

	for(i = 0; i < N; i++){
		if(i > 5){
			u[i%6] = u[(i-5)%6] ^ u[i%6];
```

```
}
if(u[i % 6] == 0) bit[i/64] <<= 1;
else{
        bit[i/64] <<= 1;
        bit[i/64] += 1;
}

x1 = u[i%6] ^ state[1] ^ state[2] ^ state[4] ^ state[5];
x2 = u[i%6] ^ state[0] ^ state[1] ^ state[2] ^ state[5];

//state shifting
for(int k = 5; k >= 0; k--){
        state[(k)%6] = state[(k-1)%6];
}
state[0] = u[i%6];

if(i != 0 && i % 64 == 0){
        j++;
        if(x1 == 1){
                cw1[j] = cw1[j] << 1;
                cw1[j] += 1;
        }
        else{
                cw1[j] = cw1[j] << 1;
        }
        if(x2 == 1){
                cw2[j] = cw2[j] << 1;
                cw2[j] += 1;
        }
        else{
                cw2[j] = cw2[j] << 1;
        }
}
else{
        if(x1 == 1){
                cw1[j] = cw1[j] << 1;
                cw1[j] += 1;
        }
        else{
                cw1[j] = cw1[j] << 1;
        }
        if(x2 == 1){
```

```
                cw2[j] = cw2[j] << 1;
                cw2[j] += 1;
            }
            else{
                cw2[j] = cw2[j] << 1;
            }
        }
    }
}

void decoder(unsigned long long int output1[], unsigned long long int Y1[], unsigned long long int Y2[],
double sigma, int mode){
    long int survivor[64][2] = {{0}, {0}};
    //check[][1] is to ensure the postition isn't go through
    //check[][2] is to store the state so that it
    int check[64][3] = {{0}, {0}, {0}};
    //metric[][0] is for storage of current distance between codeword and the trellis;
    //metric[][1] is for storage of the weight of the whole path;
    double metric[64][2] = {{0}, {0}}, m1, m2, min;
    long long int pos1 = 0, pos2 = 0, position;
    int i, j;
    unsigned long long int z = (unsigned long long)pow(2, 63);
    unsigned long long int a, b;
    long int bit = N;
    double a1, b1, a2, b2, n1, n2;
    int x1, x2, x3, x4, count = 0, state;
    //constructing table with all the states for reference
    int table[64]={0, 32, 16, 48, 8, 40, 24, 56, 4, 36, 20, 52, 12, 44, 28, 60,
                   2, 34, 18, 50, 10, 42, 26, 58, 6, 38, 22, 54, 14, 46, 30, 62,
                   1, 33, 17, 49, 9, 41, 25, 57, 5, 37, 21, 53, 13, 45, 29, 61,
                   3, 35, 19, 51, 11, 43, 27, 59, 7, 39, 23, 55, 15, 47, 31, 63};

    check[0][0] = 1;

    for(i = 0; i < N; i++){
        normal(&n1, &n2, sigma);
        if(i % 64 == 0 && i != 0) bit -= 64; //shift to the next element and minimize bit for reference of the
size of the element
        if(bit < 64) z = pow(2, bit -1); // if the next element is not fully stored, or 64 bits, extract the left
most bit

        a = Y1[i/64] & z;
```

```
b = Y2[i/64] & z;

if(mode == 1){
        //map the channel input
        if(a == 0) a1 = 1;
        else a1 = -1;
        if(b == 0) b1 = 1;
        else b1 = -1;
        //add noise
        a1 = a1 + n1;
        b1 = b1 + n2;
        //map the channel output
        if(a1 >= 0) a = 0;
        else a = 1;
        if(b1 >= 0) b = 0;
        else b = 1;
}
else if(mode == 2){
        if(a == 0) a1 = 1;
        else a1 = -1;
        if(b == 0) b1 = 1;
        else b1 = -1;

        a1 = a1 + n1;
        b1 = b1 + n2;
        //soft decision does not map the channel output
}
//start decoding...
for(j = 0; j < 64; j++){
        state = table[j];
        position = idx(table,state);

        if(check[position][0] != 0){
                //information bit equals to 0
                //Calculate metrics
                x1 = 0 ^ ((state >> 4) & 1) ^ ((state >> 3) & 1) ^ ((state >> 1) & 1) ^ (state & 1);
                x2 = 0 ^ ((state >> 5) & 1) ^ ((state >> 4) & 1) ^ ((state >> 3) & 1) ^ (state & 1);

                if(mode == 1) m1 = abs(a - x1) + abs(b - x2);
                else if(mode == 2){
                        if(x1 == 0) a2 = 1;
                        else a2 = -1;
```

```
        if(x2 == 0) b2 = 1;
        else b2 = -1;

        m1 = pow((a1 - a2), 2) + pow((b1 - b2), 2);
}
pos1 = state >> 1;
pos1 = idx(table, pos1);
//update metrics and survivor
if(check[pos1][1] == 0){
        metric[pos1][1] = m1 + metric[position][0];
        survivor[pos1][1] = survivor[position][0] << 1;
        check[pos1][2] = state;
}
else{
        if((m1 + metric[position][0]) < metric[pos1][1]){
                metric[pos1][1] = m1 + metric[position][0];
                survivor[pos1][1] = survivor[position][0] << 1;
                check[pos1][2] = state;
        }
        else if((m1 + metric[position][0]) == metric[pos1][1]){
                if(idx(table, state) < idx(table,check[pos1][2])){
                metric[pos1][1] = m1 + metric[position][0];
                survivor[pos1][1] = survivor[position][0] << 1;
                check[pos1][2] = state;
                }
        }

}
check[pos1][1] = 1;
//information bit equal to 1
x3 = 1 ^ ((state >> 4) & 1) ^ ((state >> 3) & 1) ^ ((state >> 1) & 1) ^ (state & 1);
x4 = 1 ^ ((state >> 5) & 1) ^ ((state >> 4) & 1) ^ ((state >> 3) & 1) ^ (state & 1);
if(mode == 1) m2 = abs(a - x3) + abs(b - x4);
else if(mode == 2){
        if(x3 == 0) a2 = 1;
        else a2 = -1;
        if(x4 == 0) b2 = 1;
        else b2 = -1;

        m2 = pow((a1 - a2), 2.0) + pow((b1 - b2), 2.0);
}
pos2 = (state >> 1) ^ 32;
```

```
            pos2 = idx(table, pos2);
           //update metrics & survivor
            if(check[pos2][1] == 0){
                metric[pos2][1] = m2 + metric[position][0];
                survivor[pos2][1] = (survivor[position][0] << 1) ^ 1;
                check[pos2][2] = state;
            }
            else{
                if((m2 + metric[position][0]) < metric[pos2][1]){
                    metric[pos2][1] = m2 + metric[position][0];
                    survivor[pos2][1] = (survivor[position][0] << 1) ^ 1;
                    check[pos2][2] = state;
                }
                else if((m2 + metric[position][0]) == metric[pos2][1]){
                    if(idx(table, state) < idx(table,check[pos2][2])){
                    metric[pos2][1] = m2 + metric[position][0];
                    survivor[pos2][1] = (survivor[position][0] << 1) ^ 1;
                    check[pos2][2] = state;
                    }
                }
            }
             check[pos2][1] = 1;
        }
    }
    for(j = 0; j < 64; j++){
        metric[j][0] = metric[j][1];
        metric[j][1] = 0;
        survivor[j][0] = survivor[j][1];
        survivor[j][1]=0;
        check[j][0] = check[j][1];
        check[j][1] = 0;
    }
    if(i > 30){
        unsigned long long k = (unsigned long long)pow(2, 31);
        min = metric[0][0];

        for(j = 0; j < 64; j++){
            if(metric[j][0] < min && check[j][0] != 0){
                min = metric[j][0];
                x1 = j;
            }
        }
```

```
                // determine the information bits from the survivor(the older bits are in the higher position)
                if((survivor[x1][0] & k) == 0){
                    output1[i - 31] = 0;
                }
                else {
                    output1[i - 31] = 1;
                }
            }
            //Reduce metrics every 1000 bits to prevent overflow
            if((i % 1000 == 0) && (i != 0)){
                min = metric[0][0];
                for(j = 0; j < 64; j++){
                    if (metric[j][0] < min && check[j][0] != 0){
                        min = metric[j][0];
                        x1 = j;
                    }
                }
                for(j = 0; j < 64; j++)
                    metric[j][0] = metric[j][0] - min;
            }
            Y1[i/64] = Y1[i/64] << 1;
            Y2[i/64] = Y2[i/64] << 1;
        }
}
//determine the number of erroneous bits
double BER_count(unsigned long long int output1[]){
    int u[6] = {1, 0, 0, 0, 0, 0};
    unsigned long long Y[N/64+1];
    int i, j,b = 0;
    unsigned long long a, size = N - 31;
    unsigned long long z = (unsigned long long)pow(2, 63);
    double err_count = 0;

    for(i = 0; i < N - 31; i++){
        if(i > 5){
            u[i % 6] = u[(i-5) % 6] ^ u[i % 6];
        }
        b = u[i % 6];
        if(b != output1[i]) err_count++;
    }
    return err_count;
}
```

```
int idx(int table[], int x){
	int i;
	for(i = 0; i < 64; i++){
		if(x == table[i]){
			return i;
		}
	}
}

double normal (double *n1, double *n2, double sigma){
	double x1;
	double x2;
	double s;

	do{
		x1 = Ranq1();
		x2 = Ranq1();
		x1 = 2 * x1 - 1;
		x2 = 2 * x2 - 1;
		s = x1 * x1 + x2 * x2;
	} while(s >= 1.0);

	*n1 = sigma * x1 * sqrt(-2 * log(s) / s);
	*n2 = sigma * x2 * sqrt(-2 * log(s) / s);
}

double Ranq1(void){
	if (RANI == 0){
		RANV = SEED^4101842887655102017LL;
		RANV ^= RANV >> 21;
		RANV ^= RANV << 35;
		RANV ^= RANV >> 4;
		RANV = RANV * 2685821657736338717LL;
		RANI++;
	}
	RANV ^= RANV >> 21;
	RANV ^= RANV << 35;
	RANV ^= RANV >> 4;
	return RANV * 2685821657736338717LL * 5.42101086242752217E-20;
}
```