实验九预习材料

一、 实验原理

1. RSC 码

1.1 系统码和非系统码

系统码即输出码字中既包含信息位又包含校验位，且信息位以嵌入的形式包含在输出码字流中。与系统码相反，非系统码输出的码字中不包含输入的信息位。以线性（系统）分组码为例，生成矩阵常可记作$\mathbf{G} = [I_k \ P]$，其中I_k为$k \times k$的单位阵，正是由于单位阵的存在，使得输出的码字包含信息位，如图 1

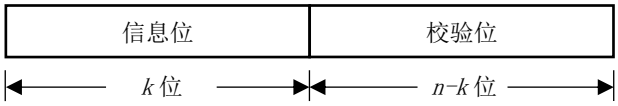

图 1 线性系统分组码

系统代码具有以下优点：校验数据可以简单地附加到信息位数据上，且如果正确接收，接收器不需要恢复原始源符号。由于上述优点，因此线性分组码通常被实现为系统码。对于卷积码，既可以被编码为系统码，也可被编码为非系统码。然而，对于某些解码算法，例如顺序解码或最大似然解码，当系统的最小汉明距离较大时，非系统结构可以在漏检译码错误概率方面提高性能。因此目前使用的实用的卷积码都是非系统码。

1.2 RSC 码的基本原理

C. Berrou 等在 1993 年提出 Turbo 码的同时，提出了一类新的递归型系统卷积（Recursive Systematic Convolutional，RSC）码，RSC 码在高码率时比最好的非系统卷积（Non-Systematic Convolutional，NSC）码还要好。一些文献已经证明：在 Turbo 码的形式下， RSC 比非递归的 NSC 具有更好的重量谱分布和更佳的误码率特性，且码率越高、信噪比越低时其优势越明显。

RSC 码可由 NSC 码转换而得到。除了 NSC 码的前馈输出，RSC 码还引入了反馈，将每一级移位寄存器的输出都反馈到输入端，并与当前输入信息位模 2 加后成为编码器输入比特，此处的反馈即为递归的思想。

每一个 RSC 码编码器，由于采用系统形式，除了前馈输出校验序列外，还会直接输出信息序列。

1.3 RSC 的编码方法

如果将生成矩阵、信息序列和码字都用二进制向量形式表示，其中码字反馈生成矢量为g_b，码字前馈生成矢量为g_f（通常将这样的生成矩阵记做$\mathbf{G} = [g_b\ g_f]$或者写成八进制形式），信息序列矢量为$\mathbf{M}$，码字矢量为$\mathbf{C}$。

图 2 所示为（2，1，3）RSC 编码器，其生成矢量为（7，5），即表示此 RSC 码编码器的码字反馈生成矢量为$g_b = [1\ 1\ 1]$，码字前馈生成矢量为$g_f = [1\ 0\ 1]$。图中⊕表示模二加法运算，在时刻i，m_i表示输入码元，a_i表示反馈输出，D1和D2表示移位寄存器，编码时初始化为零。每个时刻i，信息位x_i和校验位y_i依次输出构成 RSC 码字流c_i。

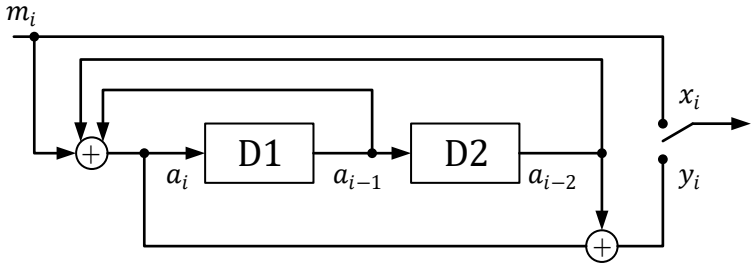

图 2 生成矢量为（7，5）的（2，1，3）RSC 码编码器

以图 2 所示的 RSC 码编码器为例，在 RSC 码的编码过程中，首先需要计算反馈输出位a_i：

$$a_i = m_i + \sum_{j=1}^{l} a_{i-j} g_{b,j+1} = m_i + a_{i-1} + a_{i-2}$$

信息输出位：

$$x_i = m_i$$

最后再计算校验输出位y_i：

$$y_i = \sum_{j=1}^{l+1} a_{i-j+1} g_{f,j} = a_i + a_{i-2}$$

最后后将x_i和y_i依次输出即构成 RSC 码字：

$$c_i = (x_i,\ y_i)$$

其中，l表示移位寄存器的个数，$g_{b,i}$表示码字反馈生成矢量g_b的第i位，$g_{f,i}$表示码字前馈生成矢量g_f的第i位，且上述公式中的加法都表示模 2 加法。

以生成矢量为（7，5）的 RSC 码为例，根据上述公式可得在时刻i，输入码元、反馈输出、信息输出、校验输出、编码输出与移存器状态的关系如表 1 所示。

表 1 输入码元、反馈输出、信息输出、校验输出、编码输出与移存器状态的关系

当前移存器 D2D1 输出 $a_{i-2}a_{i-1}$	输入码元 m_i	反馈输出 a_i	信息位输出 x_i	校验位输出 y_i	编码输出 $c_i=(x_i,y_i)$	下一时刻移存器 D2D1 输出 $a_{i-1}a_i$
00	0	0	0	0	00	00
	1	1	1	1	11	01
01	0	1	0	1	01	11
	1	0	1	0	10	10
10	0	1	0	0	00	01
	1	0	1	1	11	00
11	0	0	0	1	01	10
	1	1	1	0	10	11

定义移存器输出D2D1为 00 时表示状态 S_0，D2D1为 01 时表示状态 S_1，D2D1为 10 时表示状态 S_2，D2D1为 11 时表示状态 S_3。根据上表可以得到（2，1，3）RSC 码状态转移关系如表 2 所示。

表 2（2，1，3）RSC 码状态转移关系

当前状态	输入 0		输入 1	
	下一状态	编码输出	下一状态	编码输出
S_0(00)	S_0	00	S_1	11
S_1(01)	S_3	01	S_2	10
S_2(10)	S_1	00	S_0	11
S_3(11)	S_2	01	S_3	10

按照表 2 的规律，可以画出（2，1，3）RSC 码状态转移图如图 3 所示。状态间的连线表示它们的转移情况，线上的数字表示编码输出，线型表示编码输入：实线表示输入数据为 0 时状态转移的路线，虚线表示输入数据为 1 时状态转移的路线。

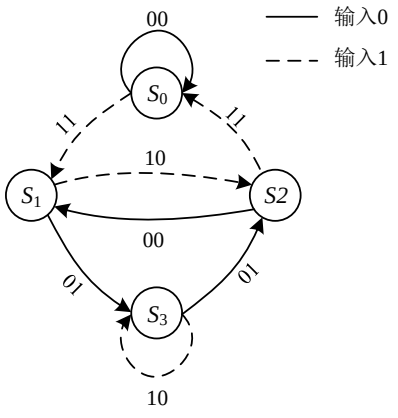

图 3 生成矢量为（7，5）的（2，1，3）RSC 码状态转移图

由于 Turbo 码译码时，需要保证 RSC 码编码器中的**最后两位反馈输出均为零状态**，当下一时刻到来时，无论输入码元为何值均可保证寄存器状态恢复为零。又因为 RSC 码编码器结构存在反馈，所以不能通过简单地在信息序列后补零实现，而需要通过计算得到待补充的拖尾比特，**需要补充l (编码器的移位寄存器个数）个拖尾比特**。对于 RSC 码，在时刻i，补充的拖尾比特必须满足下式：

$$m_i = \sum_{j=1}^{l} a_{i-j} g_{b,j+1}$$

又因为 RSC 码编码器结构要求每一个移位寄存器的输出都要有反馈连接到输入端，因此g_b一定是全 1 的状态，所以上式可以简化为：

$$m_i = \sum_{j=1}^{l} a_{i-j}$$

因此，拖尾比特m_i即为当前时刻i的所有移位寄存器的值的模 2 和。

因此，当输入的待编码信息序列$\mathbf{M} = [1\ 1\ 0\ 0\ 1\ 0\ 1\ 1]$时，按照上述过程，在当$m_7 = 1$输入后时，$\mathbf{M}$已全部输入编码器；下一时刻移存器 D2D1 的输出为$a_6 a_7 = 01$，故此时输入的拖尾比特$m_8 = 0 \oplus 1 = 1$，反馈输出为$\boldsymbol{a_8 = 0}$；当$m_8 = 1$输入后，下一时刻移存器 D2D1 的输出为$a_7 a_8 = 10$，故此时输入的拖尾比特$m_9 = 1 \oplus 0 = 1$，反馈输出为$\boldsymbol{a_9 = 0}$。当下一时刻到来时，无论输入码元为何值，下一时刻移存器 D2D1 的输出必为$a_8 a_9 = 00$，移存器的状态恢复为零。

得到的补充后的信息序列为$\mathbf{M'} = [1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1]$，对应的 RSC 码编码结果为$\mathbf{C} = [1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1]$。

2. Turbo 码编码器

Turbo 码编码器由分量编码器、交织器、删余矩阵和复接器组成，如图 4 所示。其中，分量码的最佳选择为递归系统卷积（RSC）码。

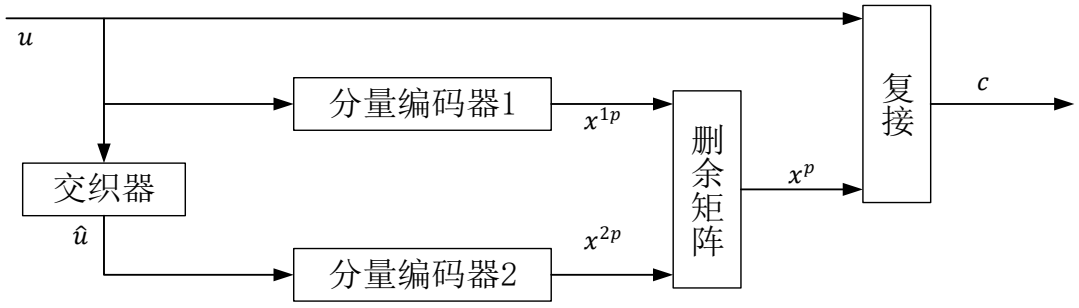

图 4 并行级联 Turbo 码的编码器结构

2.1 交织器

交织实际上就是将数据序列中元素的位置进行重置得到交织序列的过程。Turbo 码通过引入交织器，可以使得码字具有近似随机的特性。常用的几种 Turbo

码交织器有分组交织器、循环移位交织器和随机交织器等。交织器对 Turbo 码的性能基本上起决定作用，当 Turbo 码采用 RSC 码作为分量码时，Turbo 码的性能基本上和交织长度成正比。

2.2 分量编码器

Turbo 编码器中的分量码可以是 RSC 码，NSC 码或非递归卷积（Non-Recursive Convolutional，NSC）码等。其中，分量码的最佳选择是 RSC 码。通常情况下两个分量码采用相同的生成矢量得到。为保证准确译码，分量编码器 1 的输入为待编码信息序列加入拖尾比特后的序列，分量编码器 2 的输入为待编码信息序列加入拖尾比特后再经过交织器得到的序列。

在实际的 Turbo 码编码器中，RSC 编码器只输出校验位，输入码元作为信息位直接输入到复接器中。

2.3 删余矩阵

删余是通过删除冗余的校验位来调整码率。两分量码产生的校验位是对同一组信息（尽管已经交织）的校验，其数量是一个分组码的两倍。将两分量码的校验位都送到对方也非不可，但会导致码率降低，多数情况下也无必要。这时可考虑只传送其中的一部分，原则是不能完全排斥两分量码中的任何一个，而是折中地按一定规律轮流选择发送两分量码的校验比特。

举例来说，采用两个码率$\mathrm{R}=1/2$的系统卷积码时，如果不删余，信息位加两分量编码器的各一校验位将产生码率 R=1/3 的码流。但如果令编码器 1 的校验流经删余矩阵$\mathrm{P1}=[1\ \ 0]^{\mathrm{T}}$处理，而让编码器 2 的校验流经删余矩阵$\mathrm{P2}=[0\ \ 1]^{\mathrm{T}}$处理，元素‘0’表示相应位置的校验位被删除，而‘1’表示保留相应位置的校验位，那就产生了在编码器 1、2 间轮流取值的效果。此时虽然一位信息位仍然产生二位校验位，但发送到信道上的却只是一位信息位和一位轮流取值的校验位，使码率调整为$\mathrm{R}=1/2$。若编码器 1 输出的校验位码字流为$\boldsymbol{x^{1p}}=[1\,0\,1\,0\,0\,0\,1\,0\,1\,1]$，编码器 2 输出的校验位码字流为$\boldsymbol{x^{2p}}=[0\,1\,0\,1\,1\,1\,0\,0\,0\,1]$，经过删余后，删余器输出的码字流为$\boldsymbol{x^{p}}=[1\,1\,1\,1\,0\,1\,1\,0\,1\,1]$。

2.4 复接器

复接器的作用是输出系统形式的 Turbo 码的码字，它将待编码信息序列与经过删余后的序列拼接成为系统形式的码字序列。

2.5 Turbo 码编码示例

在本示例中，交织器采用随机交织器，交织长度为待编码信息序列加入拖尾比特后的长度；两个分量编码器都选择生成矢量为（7，5）的（2，1，3）RSC 码编码器；删余矩阵选择为二阶单位矩阵。本示例的 Turbo 码编码器结构如图 5 所示。

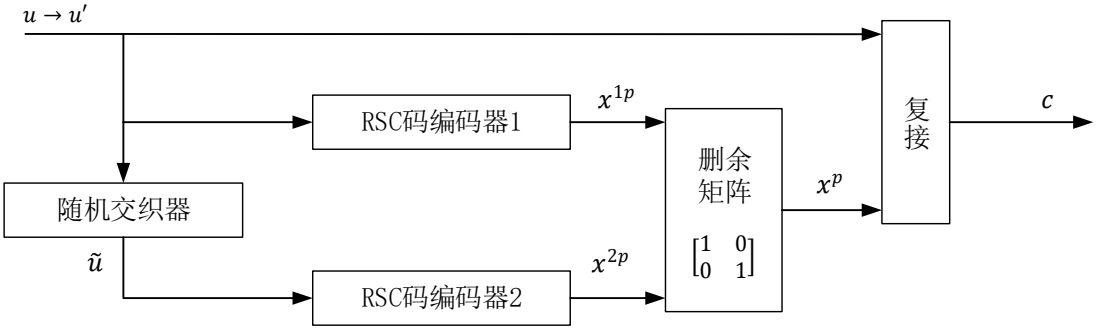

图 5 Turbo 码的编码器结构

注：图 5 中的x^{1p}和x^{2p}分别为编码器 1 和 2 输出的校验位。

假设待编码的信息序列为$\boldsymbol{u} = [1\ 1\ 0\ 0\ 1\ 0\ 1\ 1]$，由实验内容中的 RSC 码编码示例分析可知，添加拖尾比特后的信息序列$\boldsymbol{u'} = [1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1]$，其经过 RSC 码编码器 1 后并删除信息序列后（即只输出校验位）得到的输出结果为$\boldsymbol{x^{1p}} = [1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1]$。然后，将添加拖尾比特后的信息序列$\boldsymbol{u'}$经过交织长度为 10 的随机交织器得到交织后的信息序列$\tilde{\boldsymbol{u}}$，现假设随机交织器的映射表为[3 4 1 10 8 2 5 7 6 9]，因此$\boldsymbol{u'}$经过交织器后得到的交织后的信息序列$\tilde{\boldsymbol{u}} = [0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1]$，$\tilde{\boldsymbol{u}}$经过 RSC 码编码器 2 后并删除信息序列后（即只输出校验位）得到的输出结果为$\boldsymbol{x^{2p}} = [0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1]$。本示例中采用的删余矩阵DM为：

$$\mathrm{DM} = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$

即只保留$\boldsymbol{x^{1p}}$中的奇数位和$\boldsymbol{x^{2p}}$中的偶数位，并将其按索引顺序拼接在一起得到$\boldsymbol{x^p}$作为 Turbo 码的校验位。本示例中，经过删余拼接后得到校验位序列$\boldsymbol{x^p} = [1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1]$。最后将补充后的信息序列$\boldsymbol{u'}$和删余后的序列$\boldsymbol{x^p}$以$[\mathrm{u}'(1)\ x^p(1)\ \mathrm{u}'(2)\ x^p(2)\ \cdots]$的顺序拼接得到最终的系统形式的 Turbo 码码字$\mathbf{C}$，$\mathbf{C} = [1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1]$。

3. Turbo 码译码原理

(本部分了解即可，本次实验提供 Turbo 译码程序，包含 Log-MAP 和 SOVA 两种译码算法，注意：译码程序中已经包含 BPSK 解调)

3.1 Turbo 码译码结构

Turbo 码获得优异性能的根本原因之一是采用了迭代译码，通过与分量编码器对应的分量译码器之间软信息的交换来提高译码性能。对于 Turbo 码，如果分量译码器的输出为硬判决，则不可能实现分量译码器之间软信息的交换，从而限制了系统性能的进一步提高。从信息论的角度来看，任何硬判决都会损失部分信息，因此，如果分量译码器能够提供反映其输出可靠性的软信息，则其他分量译码器也可以以采用软判决译码，系统的性能可以得到进一步提高。为此，人们又提出了软输出译码的概念和方法，即译码器的输入输出均为软信息。如果交织

长度足够大，则可以把编译码器和信道一起看做等效的离散愉入无记忆广义信道，可以推得该广义信道在分量译码器采用软输出时的信道容量比硬输出时的信道容量要大。软输出译码实现了解调器和分量译码器之间的软信息转移，系统性能可以得到很大的改进。

图 6 给出了 Turbo 码的译码结构，其中每个分量译码模块均采用软输出译码算法，考虑到输入和输出均为软信息，因此用于 Turbo 码的译码算法可以称为软输入软输出（Soft Input Soft Output，SISO）译码算法。

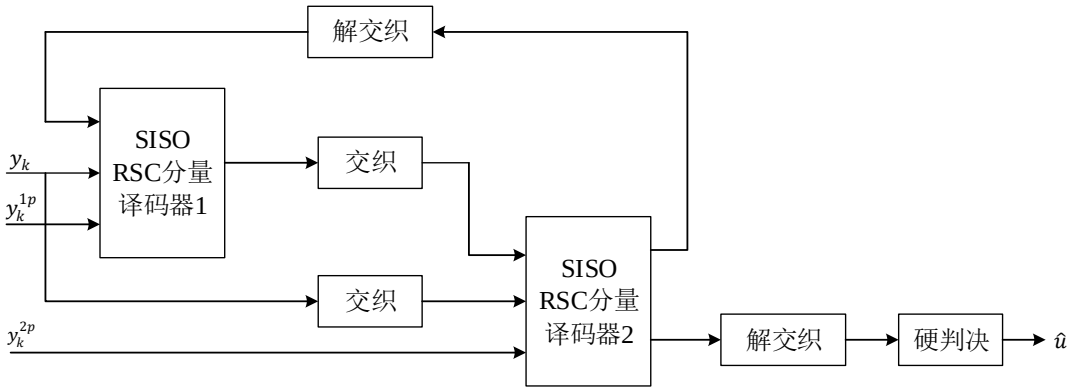

图 6　Turbo 码的译码器结构

Turbo 码译码器的基本结构如图 6 所示。 它由两个 SISORSC 分量译码器串行级联组成，交织器与编码器中所使用的交织器相同。译码器 1 对分量码 RSC1 (y_k和y_k^{1p})进行最佳译码，产生关于信息序列中每一比特的似然信息，并将似然信息经过交织后送给译码器 2，译码器 2 将此信息作为先验信息，对分量码 RSC2(y_k^{2p})进行最佳译码，产生关于交织后的信息序列中每一比特的似然比信息，然后将此似然比信息经过解交织后送回译码器 1，进行下一次译码。其中（y_k为接收的信息位，y_k^{1p}和y_k^{1p}分别为接收到的码字流）。这样，经过多次迭代，译码器 1 或译码器 2 的外信息趋于稳定，似然比渐近值逼近于对整个码的最大似然译码，经过解交织后对此似然比进行硬判决，即可得到信息序列的每一比特的最佳估值序列$\hat{u}$。显然译码性能与迭代次数有关。

3.2　Log-MAP 算法

MAP（Maximum A Posteriori）算法是基于码字网格图的软输出译码算法，目的是使译码输出比特错误概率最小。根据最大似然译码原理，译码器的主要任务就是计算在接收采样的条件下不同发送符号的概率，而后将采样接收判决为概率值最大的信息符号。

马尔科夫性是算法成立的前提和核心，任何满足马尔科夫性的编码方法，MAP 算法都使用。Log-MAP 算法与 MAP 算法等效，只是将参数转移到对数域中计算，实现较 MAP 算法简单，计算量和复杂度降低。

3.3 SOVA 算法

Viterbi 算法是卷积码译码的最优方法（等价于最大似然译码），译码输出为卷积码的最优估计序列。但对于 Turbo 码，不能直接使用 Viterbi 算法对分量码进行译码，这主要是因为：一方面一个分量译码器输出中存在的突发错误会影响另一个分最译码器的译码性能，从而使级联码的性能下降；另一方面无论是软判决 Viterbi 算法还是硬判决 Viterbi 算法，其译码输出均为硬判决信息，因此，若一个分量译码器采用 Viterbi 算法译码，则另一个分量译码器只能以硬判决结果作为输入，无法实现软输入译码，导致译码性能下降。

软输出 Viterbi 算法（Soft Output Viterbi Algorithm，SOVA）的主要思想是使 Viterbi 译码器能够提供软信息输出，并且在分量译码器之间通过软信息的交换提高 Turbo 码的译码性能。为此，需要在传统的 Viterbi 算法上进行修正，使之提供软信息输出。

SOVA 算法计算量和复杂度较 Log-MAP 算法更低，但译码性能有所下降。

二、 实验预习

生成矢量为（7，3）的（2，1，3）RSC 编码器如图 7 所示，有一待编码的信息序列矢量$\mathbf{M}=[\mathbf{1\ 1\ 0\ 0\ 1\ 0\ 1\ 1}]$，仿照实验内容中的生成矢量为(7,5)的 RSC 码编码过程表格，完成下面表 3 所示的编码过程，并写出添加拖尾比特后的信息序列以及对应的编码结果。

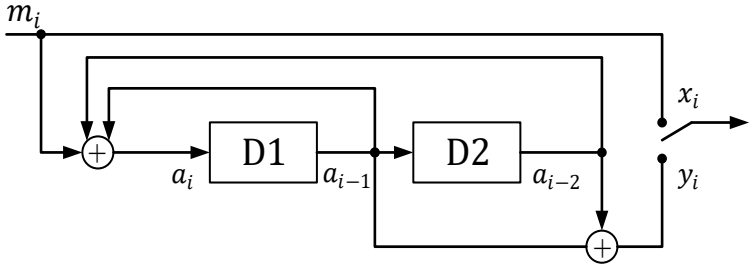

图 7 生成矢量为（7，3）的（2，1，3）RSC 编码器

表 3 生成矢量为（7，3）的（2，1，3）RSC 码编码过程

时刻 i	当前输入 m_i	当前反馈输出 a_i	当前移位寄存器 D1 的值 a_{i-1}	当前移位寄存器 D2 的值 a_{i-2}	校验位输出 y_i	编码输出 $c_i=(x_i,y_i)$
0	1	1	0	0	0	10
1	1					
2	0					
…	…					