TRANSMISIÓN DE DATOS 2006/07

Hoja de Resultados II

(Prácticas 4, 5 y 6)

A continuación se detallarán los resultados esperados para varios ejemplos significativos correspondientes a las prácticas 4, 5 y 6 de transmisión de datos. El objetivo de esta especificación de resultados es proporcionar la información necesaria para que el alumno pueda comprobar la correcta implementación de las prácticas.

1 Práctica 4: Codificación Transformacional

1.1 Ejercicio 1.

1.1.1 Ejemplo 1:

Entrada:

```
A=[87 89 101 106 118 130 142 155;
        85 91 101 105 116 129 135 149;
        86 92 96 105 112 128 131 144;
        92 88 102 101 116 129 135 147;
        88 94 94 98 113 122 130 139;
        88 95 98 97 113 119 133 141;
        92 99 98 106 107 118 135 145;
        89 95 98 107 104 112 130 144];
```

Salida:

```
dct =

  898.0000 -149.5418   26.6464  -14.0897    0.7500   -5.7540    3.5750    0.0330
   12.1982  -16.5235   -7.6122    5.2187   -0.2867   -1.9909    8.4265    1.2591
    5.3355   -2.6557    2.3410   -9.9277    2.4614    4.4558   -3.1945   -3.1640
    1.9463   -2.7271    1.5106    2.8421   -2.1336   -2.7203   -2.7510    5.4051
    0.7500   -2.0745    0.8610    0.2085    2.5000    1.8446    2.0787    2.4750
    7.9536   -2.6624    2.6308    0.4010    0.4772    3.3000    1.7394    0.3942
   -4.1042   -0.1650   -0.6945    0.0601    0.0628   -0.7874   -0.8410    0.3496
   -3.4688    2.3804    0.1559    0.8696    0.1142   -0.5240   -3.9974   -5.6187

Fint =

    56   -14     3    -1     0     0     0     0
     1    -1    -1     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

Extensión de fuente:

```
 [0,56], [0,-14], [0,1], [1,-1],[0,3], [0,-1], [0,-1],EOB

deFint =

   896  -154    30   -16     0     0     0     0
    12   -12   -14     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

```
     0     0     0     0     0     0     0     0

idct =

    84    90    99   108   117   127   139   148
    84    91    99   107   116   127   139   147
    86    91    99   106   114   125   138   147
    87    92    98   104   112   123   137   146
    89    92    97   102   110   121   135   146
    90    93    97   101   108   119   134   145
    91    93    96    99   106   118   133   144
    92    94    96    99   105   117   133   144

error =
     3    -1     2    -2     1     3     3     7
     1     0     2    -2     0     2    -4     2
     0     1    -3    -1    -2     3    -7    -3
     5    -4     4    -3     4     6    -2     1
    -1     2    -3    -4     3     1    -5    -7
    -2     2     1    -4     5     0    -1    -4
     1     6     2     7     1     0     2     1
    -3     1     2     8    -1    -5    -3     0


Perror =

  714
```

1.2 Ejercicio 2: Codificación DCT de Imágenes

1.2.1 Ejemplo 1:

Entrada:

Lenna256.bmp

Salida:
Histogramas:

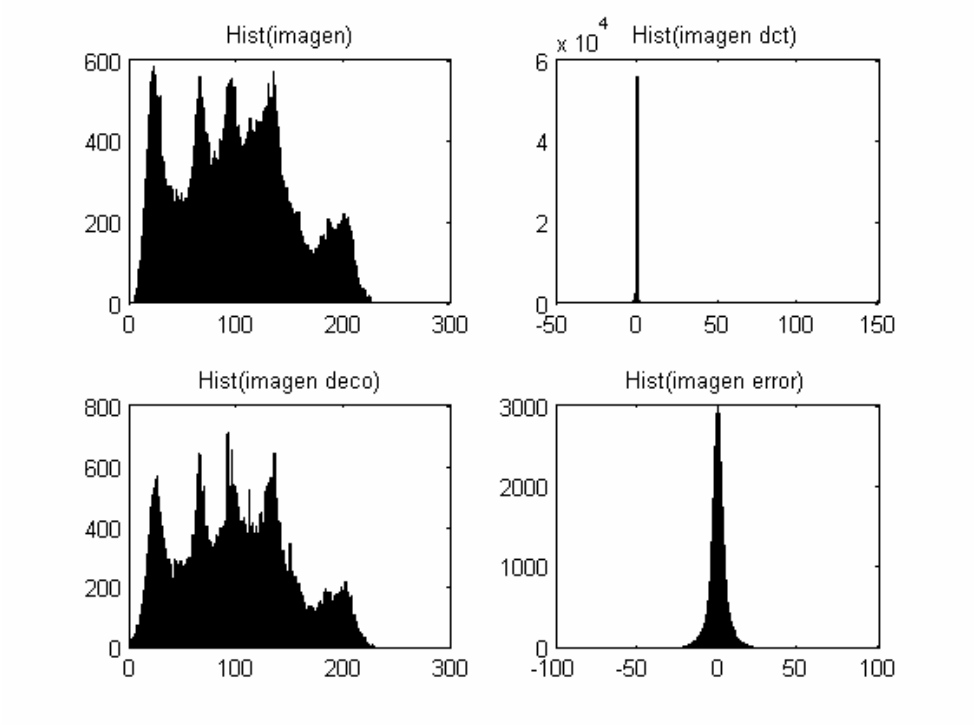

Imagen DCT cuantificada, Imagen Reconstruida, Imagen de Error

```
Perror =

   2183059
```

1.3 Ejercicio 3: Desarrollo del efecto de la cuantificación en la codificación de imágenes

Nota1: *Tenga en cuenta que la implementación requerida ha de ser en aritmética entera. Esto es, después de cada cálculo realizado, si los números resultantes son reales, habremos de convertirlos en enteros.*

Nota2: *En* `Cuantif_N.m` *realizaremos una cuantificación uniforme (*`PCMUniforme.m`*)sobre los valores del bloque de imagen.Tomaremos como entrada el bloque es si, como salida el bloque reconstruido. El valor de sobrecarga utilizado(*`v`*)será el valor absoluto máximo de los valores del bloque.*

Nota3: *En* `DCT_N.m` *realizaremos una cuantificación uniforme (*`PCMUniforme.m`*)sobre los valores de la dct(normalizada con Q) de cada bloque de imagen .Tomaremos como entrada la dct, como salida la dct reconstruida. El valor de sobrecarga utilizado (*`v`*) será el valor absoluto máximo de los valores de la dct(normalizada con Q).*

Nota4: *Para el correcto funcionamiento de PCMUniforme.m habrá de recibir como parámetro de entrada un tipo double con el fin de poder manejar valores mayores que 255.*

1.3.1 Ejemplo 1:

Entrada:

```
A =

    87    89   101   106   118   130   142   155
    85    91   101   105   116   129   135   149
    86    92    96   105   112   128   131   144
    92    88   102   101   116   129   135   147
    88    94    94    98   113   122   130   139
    88    95    98    97   113   119   133   141
    92    99    98   106   107   118   135   145
    89    95    98   107   104   112   130   144
```

Salida:

	`Cuantif_N.m`	`DCT_N.m`
Potencia de ruido (N=64)	152	287037
Potencia de ruido (N=32)	599	1148087
Potencia de ruido (N=16)	2101	4596539
Potencia de ruido (N=8)	5548	14098442

1.3.2 Ejemplo 2:

Entrada:

Lenna256.bmp

Salida:

	`Cuantif_N.m`	`DCT_N.m`
Potencia de ruido (N=64)	118525	258802708
Potencia de ruido (N=32)	434969	888509585

Potencia de ruido (N=16)	1699029	3.4432e+009
Potencia de ruido (N=8)	7212074	1.4065e+010

2 Práctica 5: Códigos Lineales

2.1 Ejercicio 1: Desarrollo de generador de códigos lineales

Matriz generatriz (5,3):

1	0	0	0	1
0	1	0	1	0
0	0	1	0	1

Matriz generatriz (7,3):

1	0	0	0	1	0	1
0	1	0	1	0	1	0
0	0	1	0	1	0	1

Mensajes/Código (5,3):

Mensaje	***Palabra código***
000	00000
001	00101
010	01010
011	01111
100	10001
101	10100
110	11011
111	11110

Mensajes/Código (7,3):

Mensaje	***Palabra código***
000	0000000
001	0010101
010	0101010
011	0111111
100	1000101
101	1010000
110	1101111

111	1111010

2.2 Ejercicio 2: Codificación lineal

2.2.1 Ejemplo:

Secuencia aleatoria:

0 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1 0 0 1

Codificación (5,3):

0 0 1 0 1

1 0 1 0 0

1 0 1 0 0

1 0 1 0 0

0 1 0 1 0

1 1 1 1 0

0 0 1 0 1

Codificación (7,3):

0 0 1 0 1 0 1

1 0 1 0 0 0 0

1 0 1 0 0 0 0

1 0 1 0 0 0 0

0 1 0 1 0 1 0

1 1 1 1 0 1 0

0 0 1 0 1 0 1

2.3 Ejercicio 3: Desarrollo de detector de errores

Matriz de chequeo de paridad (5,3):

0	1	0	1	0
1	0	1	0	1

Matriz de chequeo de paridad (7,3):

0	1	0	1	0	0	0
1	0	1	0	1	0	0
0	1	0	0	0	1	0
1	0	1	0	0	0	1

2.4 Ejercicio 4: Cálculo de síndrome

En este ejercicio las matrices de síndrome tendrán todos sus elementos iguales a cero

2.5 Ejercicio 5: Detección de errores

2.5.1 Ejemplo:

Patrón de errores aleatorio para el código (5,3):

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0

Secuencia de código (5,3) con el patrón de errores (secuencia 2.2.1 + patrón de error):

0 0 1 0 1

0 0 1 0 0

1 0 1 0 0

1 0 0 1 0

1 1 1 0 0

1 1 1 1 0

0 1 0 0 1

Patrón de errores aleatorio para el código (7,3)

0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0

Secuencia de código (7,3) con el patrón de errores(secuencia 2.2.1 + patrón de error):

0 0 0 0 1 0 1

1 0 0 1 0 0 0

1 0 1 1 1 1 0

1 0 1 0 0 0 0

0 0 0 0 0 1 0

1 1 0 1 0 1 1

0 0 1 0 1 0 1

Síndrome para la secuencia (5,3) con errores:

0 0

0 1

0 0

1 1

1 0

0 0

1 1

Síndrome para la secuencia (7,3) con errores:

0 1 0 1

1 1 0 1

1 1 1 0

0 0 0 0

0 0 1 0

0 1 0 0

0 0 0 0

2.6 Ejercicio 6: Cálculo de la matriz estándar

Matriz estándar para el código(5,3) en estudio:

[0 0 0 0 0]	[0 0 1 0 1]	[0 1 0 1 0]	[0 1 1 1 1]	[1 0 0 0 1]	[1 0 1 0 0]	[1 1 0 1 1]	[1 1 1 1 0]
[0 0 0 0 1]	[0 0 1 0 0]	[0 1 0 1 1]	[0 1 1 1 0]	[1 0 0 0 0]	[1 0 1 0 1]	[1 1 0 1 0]	[1 1 1 1 1]
[0 0 0 1 0]	[0 0 1 1 1]	[0 1 0 0 0]	[0 1 1 0 1]	[1 0 0 1 1]	[1 0 1 1 0]	[1 1 0 0 1]	[1 1 1 0 0]
[0 0 0 1 1]	[0 0 1 1 0]	[0 1 0 0 1]	[0 1 1 0 0]	[1 0 0 1 0]	[1 0 1 1 1]	[1 1 0 0 0]	[1 1 1 0 1]

Codificando el mensaje original:

0	0	1	1	0	1	1	0	1	1	0	1	0	1	0	1	1	1	0	0	1

E introduciendo el patrón de error:

```
[0     0     0     0     1
 1     0     0     0     0
 0     0     0     1     0
 0     0     0     1     1
 1     0     1     1     0
 0     0     0     0     0
 0     1     1     0     0]
```

Obtenemos el siguiente mensaje decodificado:

0	0	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1	1	0	1	0

Nota5: *A la hora de implementar la decodificación, recuerde que los primeros k bits de cada palabra código constituyen la palabra mensaje.*

Nota6: *Resulta interesante plantearse el porqué la cod-deco no corrige los errores en el ejemplo planteado. ¿Qué patrones de error sería capaz de corregir?*

3 Práctica 6: Codificación Convolucional

3.1 Ejercicio 2: Codificación Convolucional

Secuencia.txt

Mensaje	***Palabra código***
011	0 1 0 1
001	1 0 1 1
110	1 1 1 1
000	0 1 1 0
000*	0 0 0 0

Secuencia2.txt

Mensaje	***Palabra código***
000	0 0 0 0
011	0 1 0 1
001	1 0 1 1
110	1 1 1 1
000*	0 1 1 0

3.2 Ejercicio 4: Decodificación óptima por el algoritmo de Viterbi.

Secuencia3.txt

Mensaje	***Palabra código***	***Mensaje Decodificado***
01	0 1 1 0 0 1	0 1
00	1 1 1 0 1 1	0 0
11	0 1 1 1 1 0	1 1
00	0 1 0 0 0 0	0 0
00*	0 0 0 0 0 0	

Secuencia3.txt + error.txt

Mensaje	***Palabra código***	***Ruido***	***Codigo + Ruido***	***Mensaje Decodificado***
01	0 1 1 0 0 1	000100	0 1 1 1 0 1	01

00	1 1 1 0 1 1	001000	1 1 0 0 1 1	00
11	0 1 1 1 1 0	000000	0 1 1 1 1 0	11
00	0 1 0 0 0 0	000001	0 1 0 0 0 1	00
00*	0 0 0 0 0 0	000000	0 0 0 0 0 0	