Práctica 2 FECHA

Profesor: Antonio J. Caamaño Fernández. antonio.caamano@urjc.es
Profesor de prácticas: Eduardo del Arco Fernández. eduardo.delarco@urjc.es

La presente práctica está dividida en las secciones presentadas a continuación. Es necesario saber que la primera parte es un complemento de la teoría y su finalidad es garantizar una mejor y más profunda comprensión de las estructuras algebraicas que fundamentan los códigos cíclicos. La segunda parte es totalmente aplicada. Se trata de descubrir cómo funcionan los códigos de respuesta rápida o códigos QR, también llamados códigos de barras bidimensionales. Estos códigos, que forman parte de nuestras podmodernas vivencias cotidianas, hacen uso intensivo de sofisticadas técnicas de codificación en fuente y de canal. Esta parte es la que más trabajo requiere pero sus resultados son, cuanto menos, prácticos y visibles.

1. Índice

2. Breve introducción a los códigos cíclicos

Esta sección pretende ser una ayuda para construir códigos cíclicos. Carece de utilidad si no se han leido las partes propuestas del capítulo 7 del libro "Digital Communications" de John G. Proakis y las transparencias de la asignatura.

2.1. Campos de Galois

Un campo de Galois o cuerpo finito F es un conjunto finito con dos operaciones, adición $+$ y producto $\cdot$, y satisface las siguientes propiedades:

- $\{F, +, 0\}$ es un grupo Abeliano (grupo conmutativo).
- $\{F-0, \cdot, 1\}$ es un grupo Abeliano. Los elemenos no nulos del campo son grupo Abeliano para el producto, con elemento neutro 1. La inversa de $a \in F$ es a^{-1}.
- El producto es distributivo respecto a la suma: $a \cdot (b + c) = (b + c) \cdot a = a \cdot b + a \cdot c$.

Por ejemplo, $\Re$ es un cuerpo con el producto y la suma habituales, pero no es finito. El conjunto $F = 0, 1$ con suma y producto módulo 2 es un campo de Galois, un cuerpo finito. Recibe el nombre de $GF(2)$. Los campos de Galois con q elementos, llamados $GF(q)$ existen si y solo si $q = p^m$ donde p es un número primo y m es un entero positivo. Puede probarse que si $GF(q)$ existe, es único

salvo isomorfismo. Esto significa que dos campos de Galois del mismo tamaño pueden obternerse uno del otro, en el fondo lo único que estaríamos haciendo es cambiarle el nombre a los elementos. Veamos dos casos:

- Caso $q = p$, el campo de Galois se llama $GF(p) = 0, 1, 2, ..., p-1$ con suma y producto módulo p. Así, $GF(5) = 0, 1, 2, 3, 4$ es un campo finito con suma y producto módulo 5.
- Caso $q = p^m$, el campo de Galois resultante recibe el nombre de *extensión* de $GF(p)$. En este caso, $GF(p)$ es el campo base de $GF(p^m)$ y p es la *característica* de $GF(p^m)$.

Podemos definir polinómios de grado m sobre $GF(p)$:

$$g(x) = g_0 + g_1 x + g_2 x^2 + ... + g_m x^m$$

donde g_i, con $0 \leq i \leq m$, son elementos de $GF(p)$ y $g_m \neq 0$. La suma y el producto siguen son las habituales pero en módulo p. Definiciones y propiedades que se derivan:

- Si $g_m = 1$ se dice que el polinomio es *mónico*.
- Si un polinómio de grado m sobre $GF(p)$ no puede escribirse como el producto de dos polinomios de grado menor, se dice que es *irreductible*.
- Un polinomio irreductible y mónico es un *polinomio primo*.
- Un polinomio de grado m sobre $GF(p)$ tiene m raices, algunas de ellas son raices múltiples, pero las raices no "caen" necesariamente en $GF(p)$, pero sí en alguna extensión de $GF(p)$.
- Existen p^m polinomios de grado menor o igual a m, incluyendo $g(x) = 0$ y $g(x) = 1$.
- Sea $g(x)$ es un polinomio primo de grado m. Todos los polinomios de grado menor que m sobre $GF(p)$, con producto de polinomios en módulo $g(x)$ forman un campo de Galois con p^m elementos, es decir, habitan la extensión de $GF(p)$ llamada $GF(p^m) = GF(q)$.

¿Cómo es el producto módulo $g(x)$? Se toman dos polinomios de grado menor que m y se multiplican, se divide entre $g(x)$ y se toma el resto. Hagamos la operación con MATLAB:

Sea $g(x) = x^2 + x + 1$ primo sobre $GF(2)$. Se puede utilizar para construir $GF(2^2) = GF(4)$. Todos los polinomios menores de m son: $0, 1, x$ y $x + 1$. En MATLAB:

```
gf4 = gf(0:3,2)
```

Este comando genera los elementos (polinomios) $0, 1, x, x+1$ sobre $GF(2^2)$. Al ejecutarlo nos indica que el polinomio primo es $D^2 + D + 1$, que en este tutorial lo hemos llamado $g(x) = x^2 + x + 1$ y que su representación decimal es 7. Matlab denomina "array de Galois" a los elementos del campo de Galois. Las "etiquetas" de los polinomios son, poniendo a la izquierda del igual la notación decimal de MATLAB y a la derecha del igual la notación habitual:
$0 = 0$, $1 = 1$, $2 = x$, $3 = x + 1$.
Por ello el polinomio primo $x^2 + x + 1$ es [111] ó 7 en decimal. Para operar con ellos en módulo $g(x)$, simplemente hay que utilizar los operadores $+$ y $\cdot$, MATLAB se encarga de *sobrecargarlos* adecuadamente. Por ejemplo $x \cdot (x + 1)$:

```
gf4(3) * gf4(4)
```

Nos da el elemento 1, que es 1. Tomando otros polinomios $x \cdot x$:

```
gf4(3) * gf4(3)
```

Nos da el elemento 3, que en la notación normal es $x + 1$.
Nótese que todos los elementos no nulos de de $GF(4)$ pueden escribirse mediante potencias de x, en nuestro ejemplo hecho en MATLAB sería `gf4(3)`.

Para cualquier elemento no nulo $\beta \in GF(q)$, el valor más pequeño de i tal que $\beta^i = 1$ recibe el nombre de *orden* de β. En $GF(q)$, tenemos que $\beta^{q-1} = 1$. El orden de β es al menos $q - 1$. Un elemento no nulo de $GF(q)$ se llama *elemento primitivo* y su orden es $q-1$. En el ejemplo anterior, x es el elemento primitivo. Como existen muchos polinomios de grado m, existen muchas construcciones de $GF(p^m)$. Si $GF(p^m)$ está generado por $g(x)$, entonces $g(x)$ es el polinomio primitivo. El polinomio primitivo existe para cualquier grado m. Asumimos que los campos de Galois están construidos utilizando polinomios primitivos. Existen dos definiciones formales:

- Un polinomio primitivo $g(x)$ es un polinomio primo de grado m tal que si $GF(p^m)$ está contruido basándose en $g(x)$, entonces x es un elemento primitivo del campo resultante.
- Un polinomio primo de grado m $g(x)$ es primitivo si $g(x)$ no divide a $x^i + 1$ para cualquier $i < p^m - 1$.

El *polinómino mínimo* de un elemento de $GF(p^m)$ es el polinomio mónico de menor grado sobre el campo base $GF(p)$ que tiene a ese elemento como raiz. Sea β un elemento no nulo de $GF(2^m)$. El polinomio mínimo de β, llamado $\phi_\beta(x)$, es un polinomio mónico de menor grado con coeficientes en $GF(2)$ tal que β es raiz de $\phi_\beta(x)$. Es decir, $\phi_\beta(\beta) = 0$. $\phi_\beta(x)$ es primo sobre $GF(2)$. Intentemos comprender esto con un ejemplo en MATLAB:

```
m = 4
e = gf(6,m)
em = minpol(e)
emr = roots(gf([0 0 1 1 1],m))
```

2.2. Construyendo un código lineal con un polinomio generador sobre un campo de Galois

Empezamos con un ejemplo sencillo: Considere el código polinómico construido sobre $GF(2^m) = GF(2)$ (es decir, $m = 1$) y con $n = 5$. Considere el polinomio generador $g(x) = x^2 + x + 1$, de grado $r = 2$. El grado del polinomio determina el número de palabras código, es decir, 2^{n-r} y son las siguientes palabras:

$$0 \cdot g(x)$$
$$1 \cdot g(x)$$
$$x \cdot g(x)$$
$$(x + 1) \cdot g(x)$$
$$x^2 \cdot g(x)$$
$$(x^2 + 1) \cdot g(x)$$
$$(x^2 + x) \cdot g(x)$$
$$(x^2 + x + 1) \cdot g(x)$$

Desarollándolo:

$$0$$
$$x^2 + x + 1$$
$$x^3 + x^2 + x$$
$$x^3 + 1$$
$$x^4 + x^3 + x^2$$
$$x^4 + x^3 + x + 1$$
$$x^4 + x^3 + x + 1$$
$$x^4 + x^2 + 1$$

Expresándolo en dígitos binarios, las palabras código son:

00000
00111
01110
01001
11100
11011
10010
10101

Nótese que es un código lineal: Combinaciones lineales de palabras código son de nuevo palabras código. Así: $00111 + 10010 = 10101 \bmod 2$. Para construir las palabras código a partir de las palabras dato utilizamos un método sistemático. Sea una palabra dato $d(x)$ de longitud $k = n - r$, multiplicamos $d(x)$ por x^r, que en $GF(2)$ es equivalente a desplazar la palabra dato $d(x)$ r lugares a la izquierda. En general, $x^r \cdot d(x)$ no será divisible entre $g(x)$, es decir, no será una palabra código válida. Para obtener palabras código válidas, computamos el resto dividiendo $x^r \cdot d(x)$ por $g(x)$:

$$x^r \cdot d(x) = g(x) \cdot q(x) + r(x)$$

Donde $r(x)$ tiene grado menor que r. La palabra código correspondiente palabra dato $d(x)$ se define:

$$p(x) := x^r \cdot d(x) - r(x)$$

Y tiene las siguientes propiedades:

- $p(x) = g(x) \cdot q(x)$ es divisible entre $g(x)$.
- Como $r(x)$ es de grado menor que r, los primeras $k = n - r$ símbolos de la palabra código válida es palabra de datos.

El código que hemos construido no es cíclico, con lo cual no es BCH ni RS. Al menos es lineal, sistemático y se construye mediante un polinómio generador.

2.3. Construcción de un código cíclico

Un código cíclico es un código que cumple la siguiente propiedad: Sea $\boldsymbol{c} = (c_{n-1}, c_{n-2}, ..., c_1, c_0)$ una palabra código de un código cíclico, entonces $(c_{n-2}, c_{n-3}, ..., c_1, c_0, c_{n-1})$ también es una palabra código. Para que el código sea cíclico el polinomio generador debe ser divisible entre $x^n + 1$. Para un desplazamiento cíclico de i posiciones, se obtiene:

$$p(x)^(i) = x(i)p(x)mod(x^n + 1)$$

Expresión que ha sido demostrada en la asignatura, (transparencias tema 8, páginas 8-13). Veamos otro ejemplo: Considere un código con palabras de longitud $n = 7$. Empezamos descomponiendo en factores el polinomio $x^7 + 1$:

$$x^7 + 1 = (x + 1)(x^3 + x^2 + 1)(x^3 + x + 1)$$

Como el polinomio generador $g(x)$ ha de ser divisible entre $x^7 + 1$, los candidatos son cualquiera de los factores de $x^7 + 1$. Elegiremos aquel que más nos convenga. En un código (n, k) elegimos el de grado $r = n - k$. En este caso hay dos posibilidades:

$$g_1(x) = x^3 + x^2 + 1$$
$$g_2(x) = x^3 + x + 1$$

Estos códigos son equivalentes, tienen la misma capacidad correctora, el mismo número de símbolos, etc. Las palabras código son, expresadas en dígitos binarios:

0000000
0001101
0011010
0010111
0110100
0111001
0101110
0100011
1101000
1100101
1110010
1111111
1011100
1010001
1000110
1001011

Es evidente que no podemos diseñar un código cíclico para cualquier par (n, k). Para que el código cíclico sea sistemático, es necesario:

1. Multiplicar la palabra dato en forma polinómica $d(x)$ por x^r.
2. Dividir $x^r d(x)$ por $g(x)$ para obtener el resto $r(x)$.
3. Añadir $r(x)$ a $x^r d(x)$.

2.4. Algunos códigos cíclicos interesantes

A continuación repasaremos algunos códigos cíclicos interesantes:

1. **Códigos cíclicos de Hamming**, con $(n, k) = (2^m - 1, n - m)$, es decir $r = m$ ó tienen m bits de paridad.

2. **Códigos cíclicos de Golay**, el código lineal $(n, k) = (23, 12)$ se genera con el polinomio $g(x) = x^1 1 + x^9 + x^7 + x^5 + x + 1$. Las palabras código tienen una distancia mínima $d_{min} = 7$.

3. **Códigos cíclicos de máxima longitud**, son códigos duales a los Hamming, eso significa que $(n, k) = (2^m - 1, m)$. Estos códigos existen para cualquier valor positivo de m. Cada palabra código, a excepción de la todo ceros, contiene 2^{m-1} "unos" y $2^{m-1} - 1$ "ceros". La distancia mímina es $d_{min} = 2^{m-1}$.

Los códigos cíclicos más importantes son los BCH, Bose-Chaudhuri-Hocquenghem. Los códigos BCH incluyen códigos para alfabetos binarios y no binarios. Los códigos BCH pueden decodificarse de forma algebraica mediante decodificación por síndrome. Son códigos bien conocidos y están perfectamente parametrizados para distintos tamaños de bloque y tasas de codificación. De hecho están considerados la mejor familia conocida de códigos para tamaños de palabra de pequeño a mediano. Un código BCH satisface las siguientes relaciones:

$$n = 2^m - 1$$
$$n - k \leq mt$$
$$d_{min} \geq 2t + 1$$

La primera expresión determina la longitud del código. La segunda expresión proporciona una cota del chequeo de paridad en función de su capacidad correctora. La tercera expresión nos indica que el código corrige al menos t errores. Para más información, consúltense las páginas 463 a la 471 del libro "Digital Communications" de J. Proakis

3. Introducción a los códigos Reed-Solomon

Los códigos Reed-Solomon, RS a partir de ahora, pertenecen a la familia de códigos cíclicos correctores de errores. Fueron inventados en 1960 por Irving S. Reed y Gustave Solomon, miembros en aquella época del Lincoln Laboratory del MIT. Los códigos RS describen una forma sistemática de construir códigos que pueden detectar errores múltiples en símbolos. Las características principales de los códigos Reed-Solomon son:

1. Un código RS es un código FEC. Tal como se ha visto en la asignatura "Transmisión Digital", se añade redundancia a la información transmitida utilizando un algoritmo. Los códigos FEC o "corrección de errores hacia delante" permiten detectar errores y corregirlos sin necesidad de retransmisión de la información.

2. Un código RS es un código bloque lineal.

3. Un código RS es un código cíclico.

4. Un código RS es un código BCH.

5. Los códigos RS (n, k) funcionan con k símbolos que consisten en m bits que se codifican para generar palabras código de n símbolos. Para estos códigos $n = 2^m - 1$, el número de símbolos de chequeo de paridad $n - k = 2t$ y $d_{min} = 2t + 1$. Es importante hacer notar que los símbolos están formados por m bits, por lo tanto...

6. Los símbolos del código son coeficientes de un polinomio en un campo de Galois $GF(2^m)$.

7. Un código RS es un "erasure code" óptimo o código óptimo con borrado. En un código de este tipo sólamente es necesario un subconjunto de k símbolos de la palabra de n símbolos para recuperar el mensaje original.

8. Un código RS es un código de máxima longitud. Para cada entero $k \geq 3$ existe un código de máxima longitud (n, k) con $n = 2^k - 1$ y $d_{min} = 2^{k-1}$

Una palabra dato para un código RS (n, k) consiste en $m \cdot k$ bits, que lo interpretaremos como si fueran k símbolos entre 0 y 2^m. Estos símbolos son secuencias binarias de longitud m, que se corresponden con los elementos del campo de Galois $GF(2^m)$, en orden descendente. En esta práctica utilizaremos un formato de números enteros para los códigos RS. Esto permite estructurar los mensajes y palabras código como números enteros en vez de señales binarias. Tiene dos ventajas:

1. Resulta más sencillo de leer. Cuando trabajamos con muchos símbolos, por ejemplo, una modulación 64-QAM, representamos los símbolos de seis bits con un número entre 0 y 63.

2. Permite utilizar generadores de enteros aleatorios. En MATLAB existen funciones con buenas prestaciones para esta finalidad. En esta práctica se utilizará una fuente de números aleatorios uniforme.

4. Desarrollo de la práctica

Los códigos RS son muy útiles y se utilizan masivamente. El caso más simple, que es el que estudiaremos en la presente práctica, la longitud de palabras código es $n = 2^m - 1$, donde 2^m es el número de símbolos del código. La capacidad de correción es $\lfloor \frac{n-k}{2} \rfloor$. $n - k$. La práctica consta de dos partes:

1. Se pondrán en práctica algunos conceptos sobre campos de Galois con ayuda de MATLAB.

2. Se estudiará un sistema real de codificación de canal, los códigos QR.

4.1. Parte 1: Campos de Galois

1. Arranque MATLAB.

2. Hemos contruido todos los elementos de $GF(2^2)$. Utilizando los comandos aprendidos en la introducción, construya $GF(2^3) = GF(8)$ con polinomio primitivo (primo, generador) $g(x) = x^3 + x + 1$ y responda a las siguientes preguntas:

   a) Enumere los elementos de $GF(8)$.

b) Construya la tabla de multiplicar de $GF(8)$.

c) Encuentre los elementos primitivos de $GF(8)$.

d) Ejecute la función `minpol` sobre el campo $GF(8)$ completo. Proporcione una interpretación del resultado.

3. Construya, con ayuda de las funciones `gf`, el código lineal presentado en la sección 2.2.

4. Construya, con ayuda de las funciones `gf`, un código cíclico presentado en la sección 2.3.

4.2. Parte 2: Codificación y decodificación de un código QR

Los códigos QR o "Quick Response" fueron inventados por Denso, filial de componentes de Toyota, a mediados de los años 90 del siglo pasado. Antes de la aparición de los pequeños computadores de bolsillo que son los teléfonos actuales, decodificar un código QR requería un lector específico, por lo tanto estaban limitados a sus ámbitos industriales y logísticos originales. Con la popularización de los "smartphones" con cámara de fotos y aplicaciones de terceros, los códigos QR empezaron a verse por todas partes.

En esta práctica se ha llevado a cabo una realización parcial del estándar "ISO/IEC 18004. Information technology — Automatic identification and data capture techniques — Bar code symbology — QR Code". Primero una demostración: ejecute el script `p2_script.m` tal como se presenta y observe el resultado. Si dispone de un "smartphone", descargue una aplicación para leer estos códigos y trate de leer el resultado. La ejecución del código también ha dado lugar a diversas variables y mensajes en la pantalla principal. Antes de introducirse de lleno en los detalles de la codificación, vamos a ensayar algunas variables del "script":

1. Ensaye distintos textos y longitutes. Observará que la longitud está limitada y tampoco puede utilizar acentos y otros caracteres propios del Español. También observará que la longitud de la palabra dato y la palabra código no cambian con la longitud del texto, siempre es la misma.

2. Ensaye distintos valores para ECL dentro del rango indicado. Observará que la longitud del texto disminuye si aumentamos ECL, la palabra dato disminuye su longitud y la palabra código la mantiene. La palabra código es un bloque RS (N, K, R), donde N es la longitud de la palabra código, K es la longitud de la palabra dato y R es la redundancia. N es 26 en cualquier caso, ya que este código QR tiene una carga útil de 208 bits ó 208 módulos. Por lo tanto, cuando aumentamos ECL lo que hacemos es disminuir K y aumentar R.

3. Intente decodificar el QR con su móvil para diferentes ECL. En este momento no se están produciendo fallos, así que debería funcionar rápidamente. Ponga el puntero del ratón en la parte central del QR y cambie el tamaño de la ventana. El puntero afectará a más o menos módulos. Observe que a partir de cierta desaparición de módulos, el QR no se puede decodificar si ECL no es lo sufientemente elevado. Acaban de hacer un pequeño experimento de borrado. El decodificador del móvil conoce qué módulos han desaparecido. Si el número de módulos excede las capacidades del código, ni siquiera intenta decodificarlo.

4. Cambie la variable "id" y observe los QR resultantes. Son diferentes y todos válidos. Esto es debido al uso de máscaras 8 máscaras diferentes. Las máscaras sirven para romper patrones que se pudieren formar en el diseño del QR y facilitar la detección de los módulos. La elección de una máscara u otra ha de ser evaluada de acuerdo con una serie de reglas muy precisas definidas en el estándar.

5. Observe que la función `qr_channel` no introduce perturbaciones en el canal. Más adelante se pedirá utilizar el prototipo de la función para introducir diferentes perturbaciones.

Una vez ensayados algunas opciones en el "script" inicial, es el momento de profundizar en los códigos QR de la mano del estándar. El documento es muy amplio y es sencillo perderse. Con la siguiente guía podrá consultar las secciones exactas del estándar. Para adquirir el lenguaje necesario, lea las páginas 2, 3, 4 y 5 del estándar (documento adjunto). Edite el archivo `qr_encode.m` y siga

los siguientes puntos junto a las páginas que se indican del estándar. Los comentarios en el código le permitirán relacionar la presente guía con el código y con el estándar.

1. La presente realización genera códigos QR versión 1, es decir, con un tamaño de 21x21 módulos. En las páginas 6 a la 12 puede ver diferentes tamaños, desde la versión 1 a la versión 40. Los módulos están agrupados en pequeños rectángulos de 8 bits y cada rectángulo es un símbolo RS. Los bits están plegados en "zigzag" dentro de cada símbolo. En las páginas 46 a la 49 se explica cómo se sitúan los símbolos en el QR y los diferentes plegados. Un detalle importante es que, en la versión 1, se utiliza un sólo bloque RS de 26 símbolos. Versiones más altas utilizan varios bloques que se disponen de forma entrelazada. En la página 15 encontrará una tabla con capacidades según versión.

2. En los códigos QR hay una serie de módulos que son fijos. Se trata de los patrones de búsqueda, alineación y tiempo. La descripción de éstos se encuentra en la página 13.

3. Antes de hacer la codificación RS y colocar los símbolos en el código QR se realiza una codificación en fuente. El estándar contempla diferentes codificaciones de fuente descritas a partir de la página 16. La presente realización codifica el texto siguiendo el modo alfanumérico descrito en la página 21. Básicamente, dos caracteres consecutivos se codifican en 11 bits y un caracter aislado se codifica en 6 bits, mediante el diccionario de la tabla 5. Es necesario añadir una cabezera indicando el modo y el número de caracteres. Por último, es necesario añadir un terminador que consiste en 4 ceros y símbolos de relleno hasta completar la capacidad total del QR (ver página 27, secciones 8.4.8 y 8.4.9).

4. Las tablas 7 a 11 encontrará las capacidades en caracteres dependiendo de la versión, del ECL y el tipo de codificación en fuente.

5. La capacidad de detección y corrección de errores y borrados depende del ECL. En la página 33 encontrará estas capacidades y en la tabla 13 a 22, el tipo de código RS utilizado según ECL y versión.

6. Para generar la palabra código se utiliza el circuito de la página 45. Las operaciones se hacen en $GF(2^8)$. Los polinómios generadores están en el anexo A, en la página 67.

7. Tal como se ha explicado con anterioridad, es necesario romper los posibles patrones para lo cual se establecen 8 máscaras diferentes. La realización actual genera las 8 máscaras con la función `masking.m`. Las máscaras están descritas en las páginas 50 y 51. Las reglas para elegir una máscara están descritas en la página 52.

8. El código QR dispone de unas zonas reservadas con información sobre el formato. Se trata de una secuencia de 15 bits con codificación BCH $(15, 5)$. La construcción de la palabra código de formato está descrita en la página 53 y la colocación de esta palabra código, que se hace por duplicado, en la página 54.

9. A partir de la versión 9, se incluye información sobre la versión del QR, y por lo tanto sobre su tamaño, en un campo aparte. Como se ha realizado la versión 1, esta sección puede ser ignorada.

10. A partir de aquí disponemos de los ingredientes necesarios para fabricar códigos QR versión 1, con todas las ECLs posibles, en modo alfanumérico.

Una vez comprendido cómo funciona, responda a las siguientes preguntas:

1. ¿Qué ventaja tiene aplicar un sistema de corrección de errores RS, que se aplican sobre símbolos completos de 8 bits, en códigos QR?
2. ¿Por qué se entralazan los bloques RS en versiones elevadas?
3. A la vista de `qr_encode.m`, explique paso a paso cómo se elabora la palabra dato, haciendo especial énfasis en los caracteres de relleno. Ensaye diferentes textos y longitudes si es necesario.
4. Explique cómo se elabora la palabra código RS. ¿Qué tipo de codificador es? ¿Por qué decimos que es un código acortado?
5. De acuerdo con la información presentada en el estándar (pagina 33) ¿Cuál es la capacidad correctora de nuestro código?
6. En el estándar se propone un procedimiento de decodificación y se hacen una serie de recomendaciones, sobre todo para la parte de detección de los patrones y módulos. A la vista de `qr_payload_decode.m`, explique el procedimiento de decodificación tal como se ha implementado en ésta práctica.

Para la finalización de la práctica, se pide:

1. Una codificación de canal tan potente permite corregir fallos por la *vandalización* de un código QR. Tomando como modelo el prototipo de la función `qr_channel`, simule un canal binário simétrico con probabilidad de cruce p. Analice las prestaciones de la codificación frente a este tipo de canal.
2. Simule un canal con borrado en los símbolos. Para ello tiene que habilitar el puerto de borrado en el codificador. Consulte la ayuda de MATLAB de la función `fec.rsdec`. Muestre que las prestaciones del código son, las que promete el estándar.

Si quiere optar a una generosa cantidad de puntos extra:

1. Proponga un algoritmo de colocación de los símbolos para elaborar QR hasta versión 9 sin tener que hacer un mapeo manual de los símbolos.