# Comunicaciones Inalámbricas

Laboratorio 1 - v.2022.1

Repaso Modulación Digital Lineal

## 1. Introducción

En este laboratorio partiremos de un par transmisor/receptor digital BPSK, como el que se analiza en la asignatura Comunicaciones Digitales, para transformarlo en un modulador QPSK. Además, exploraremos con algo de cuidado la elección del pulso. La idea será repasar algunos de los resultados y fenómenos que se vieron en asignaturas previas. El diagrama completo del sistema a simular se puede observar en la figura 1.

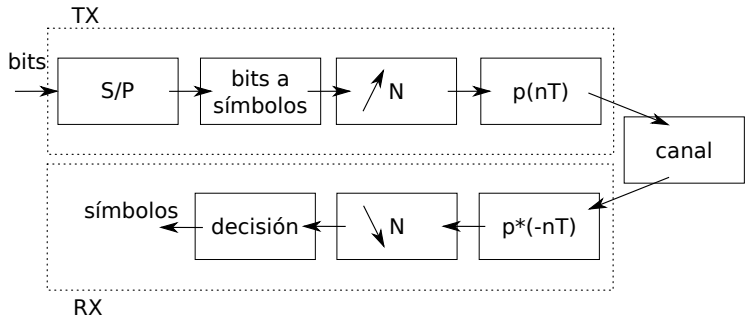


Figura 1: El sistema a simular en el obligatorio.

A continuación se discuten los cambios necesarios respecto al sistema BPSK que se entrega junto con esta letra.

## 2. Transmisor

- **La fuente de bits.** Es importante tener claro que en GNU Radio existen tres tipos de datos que "equivalen" a bits: `Byte`, `Char` y `Unsigned Char` (`UChar`). El tipo `Char` es un byte con signo, cuyo valor numérico va entre -128 y 127 (aritmética complemento a 2). El tipo `UChar` es un caracter sin signo, por lo que su valor va entre 0 y 255. Por último, `Byte` se puede considerar como exactamente lo mismo que `UChar`. Tener esto claro será importante a la hora de convertir entre tipos. Por ejemplo, no existe un conversor entre tipo `Float` y `Byte`, pero sí entre `Float` y `UChar`.

  Respecto a cómo generar una sucesión de bits, si quisiéramos generarlos de manera aleatoria, podríamos usar un bloque `Random Source` con salida del tipo

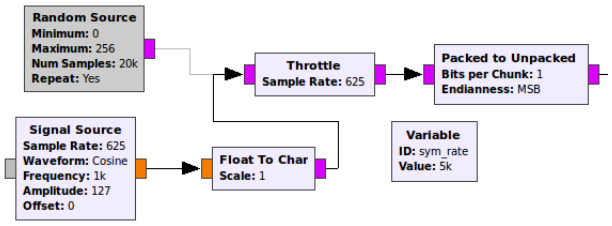


Figura 2: Primer etapa: generación de bits y bloque S/P en BPSK

`Byte` y que vaya entre 0 y 255. Generar números aleatorios por lo general es costoso y enlentecerá la ejecución. Para evitar esto se puede usar una fuente como la `Signal Source` con una amplitud de 127, seguida de un conversor `Float to Char`.

- **El bloque Serie a Paralelo (BPSK).** Lo que generamos en el paso anterior fue un tren de bytes. Si necesitamos generar un tren de a 1 bit (dado que utilizaremos únicamente dos formas de onda), esta tarea la hace el bloque `Packed to Unpacked`. El argumento de este bloque es `Bits per Chunk`, que en este caso fijaremos en 1. A manera de ejemplo, si a este bloque le ingresa el byte `00011011` sacará los siguientes ocho bytes: `00000000`, `00000000`, `0000000`, `0000001`, `00000001`, `00000000`, `0000001`, y `0000001`. Por tanto, en este caso la tasa de salida es ocho veces mayor a la de entrada. Será ésta la tasa de generación de símbolos.

  El diagrama resultante de esta primera etapa en GNU Radio para BPSK se puede apreciar en la figura 2. Se muestran las dos opciones de generación de bits: aleatorio y con una fuente de señal. Notar que se asignó una variable `sym_rate` a 5000 como tasa de generación de símbolos, y que el `Sample Rate` se fijó acorde a lo discutido.

- **El bloque Serie a Paralelo (QPSK).** Al utilizar QPSK, cada símbolo será ahora de dos bits. Habrá que modificar entonces el bloque `Packed to Unpacked` y en particular su parámetro `Bits per Chunk`, que en este caso fijaremos en 2. Por tanto, en este caso la tasa de salida es cuatro veces mayor a la de entrada. Será ésta la tasa de generación de símbolos, si queremos mantener esta tasa con valor 5000, ¿qué corresponder ingresar como `Sample Rate` en los bloques `Throttle` y `Signal Source`?

- **Mapeo a símbolos.** En este ejemplo usaremos la constelación QPSK estándar que aparece en la figura 3.

  ¿Qué corresponde ingresar como `Symbol Table` en este caso?

  Le recordamos que necesita ingresar una lista entre paréntesis rectos, con valores separados por coma. A su vez, el número imaginario $j$ se escribe `1j`.

- **Upsampling y conformador.**

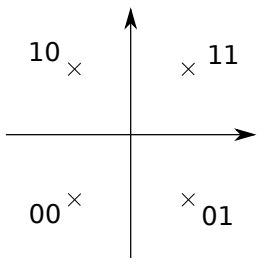


Figura 3: La constelación QPSK elegida.

Imaginemos que queremos enviar un único bit. Entonces deberíamos transmitir por el canal la señal $a\phi(t)$ (siendo $a$ el símbolo correspondiente al bit que queremos enviar). En este caso, que enviamos un bit cada $T_b$ segundos, y tal como lo indica la intuición, deberíamos enviar por el canal la señal $s(t) = \sum_k a_k \phi(t - kT_b)$ (i.e. la forma de onda correspondiente al $k$-ésimo bit es enviado en el instante $kT_b$).

Cualquier bloque interpolador de GNU Radio (como el `Polyphase Interpolator` o el `Interpolating FIR Filter`) incluyen la posibilidad de asignar un filtro arbitrario a la salida del bloque de upsampling. Para implementar la etapa de *conformación* en GNU Radio utilizaremos el bloque `Polyphase Interpolator` (ver figura 4). La salida de este bloque será $s[n] = \sum_k a_k \phi(nT - kT_s)$, donde las muestras $\phi(nT)$ se indican en el argumento `Taps`.

Es importante resaltar dos cosas. Primeramente, estamos simulando un sistema continuo por uno discreto, tomando muestras cada $T$ segundos. Para fijar el valor de $T$, definiremos una variable `samp_per_sym` que indicará cuántas muestras se tomarán por símbolo (o bit en este caso). Por lo tanto, se cumplirá la siguiente igualdad $T = T_b/$`samp_per_sym`. Note que ahora existen dos rates: la tasa de símbolo (`sym_rate`) y la tasa de muestras (`samp_rate`), siendo esta última igual a `sym_rate*samp_per_sym`. Para ser conservadores, fijaremos `samp_per_sym` en 10. Es precisamente este valor el que hay que ingresar en el campo `Interpolation` del bloque. Tomar en cuenta además, que de aquí en adelante la tasa de la señal se multiplicó por 10.

El segundo aspecto a discutir es cómo se fijan las muestras de $\phi(t)$. Para ingresarlas al bloque basta con poner la lista de valores numéricos $\phi(nT)$ entre paréntesis rectos.

Para generar el pulso rectangular podemos asignar a la variable `pulso_rect` el valor `list([1])*samp_per_sym`.

¿Qué corresponde ingresar si $\phi(t)$ fuera un pulso Manchester? Guarde los taps en la variable `pulso_manchester`

Verifique que su implementación del nuevo pulso está funcionando correctamente visualizando la señal generada en el tiempo. Para ello puede usar un bloque `QT GUI Time Sink`.

Ahora experimentaremos con el pulso Squared Root Raised Cosine (SRRC). Los taps correspondientes a cada filtro los guardaremos en la variable `pulso_srrc`.

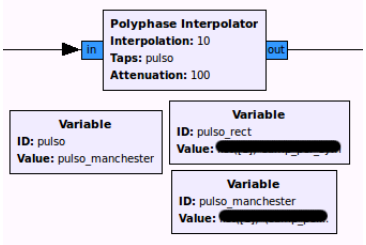


Figura 4: Tercera etapa: conformador de pulsos. La variable `samp_per_sym`, no mostrada, se utilizó para fijar el valor de `Interpolation`. Notar la existencia de la variable `pulso` para elegir entre los dos pulsos. La censura al pulso Manchester fue inevitable.

Generar el pulso SRRC será un poco más complicado. Por lo pronto habrá que incluir un bloque `Import`, que incluirá la línea `from gnuradio.filter import firdes`. Esto nos habilitará a llamar al método `firdes.root_raised_cosine`. La documentación de este método indica lo siguiente: `root_raised_cosine(double gain, double sampling_freq, double symbol_rate, double alpha, int ntaps)`.

Si la variable `pulso_srrc` la asignáramos con el comando `firdes.root_raised_cosine(1, samp_rate, samp_rate/samp_per_sym, 0.5, 5*samp_per_sym)`, ¿qué caractéristicas tiene este pulso?

La idea ahora será generar pulsos con distintas características, por lo que tendrá que agregar un par de variables indicando la cantidad de símbolos que quiere que ocupe el pulso, y su exceso de ancho de banda ($\alpha$). Es importante remarcar que el pulso SRRC necesariamente tiene que tener una cantidad impar de muestras, por lo que tanto las muestras por símbolo (`samp_per_sym`) como el largo en símbolos del pulso (referenciado más adelante como `len_sym_srrc`) deben ser impares.

Verifique que su implementación está funcionando correctamente visualizando la señal generada en el tiempo (por ejemplo con un bloque `QT GUI Time Sink`). Luego analice el espectro (por ejemplo con un bloque `QT GUI Frequency Sink`).

¿Qué espectro tiene la señal cuando es conformada con un pulso rectangular? ¿Y con un pulso SRRC? En este último caso tome distintos valores de $\alpha$ y de largo del pulso. ¿Se cumple que el ancho de banda de la señal es igual a $(1+\alpha)/(2T_s)$?

# 3. Canal

En este laboratorio el canal será simulado con un simple "pasa-todo". De todas formas, utilizaremos el bloque `Channel Model` para irnos familiarizándonos con él. Estudie con cierto detenimiento el bloque y asigne la potencia del ruido, el offset en frecuencia y la diferencia entre relojes en 0 (para lograr esto último hace falta asignar el parámetro `Epsilon` en 1). Además, dado nuestro escenario ideal, el parámetro `Taps` será simplemente 1.

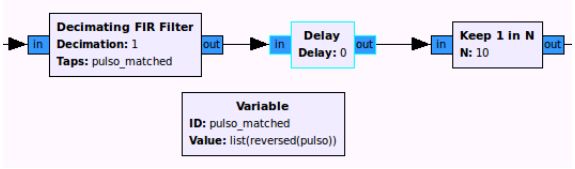


Figura 5: Primeros dos pasos del receptor: filtro apareado y muestreador. Notar la presencia de la variable `pulso` definida antes y la manera en que generamos la respuesta apareada.

# 4. Receptor

- **Filtro apareado.** Como en este caso no tenemos problemas de canal, el primer paso será pasar la señal recibida por el filtro apareado. Es decir, por un filtro con respuesta a impulso el pulso que hayamos elegido en transmisión invertido en el tiempo ($\phi(-t)$, sin el conjugado dado que estamos trabajando con reales). Con este fin podemos usar un bloque `Decimating FIR Filter`, con parámetro `Decimation` fijado en 1 (i.e. sin decimar, pues el muestreo lo haremos con cuidado más adelante) y con `Taps` igual al pulso que hayamos elegido, a excepción de que ahora están ordenadas al inverso (i.e. las muestras $\phi(-nT)$). Aquí ilustraremos el uso un poco más avanzado de sentencias Python dentro del Companion, pues podemos hacer esta inversión en una sola línea utilizando `list(reversed(pulso))` (ver figura 5). Busque la documentación de estas dos funciones Python para ver qué hacen cada una.

  ¿Qué espera ver a la salida del filtro apareado cuando utiliza el pulso rectangular? Verifique que su implementación es la correcta comprobando que ve lo que se espera.

- **Muestreo.** Como todavía no hemos visto las técnicas de corrección del timing, en este laboratorio implementaremos una técnica "artesanal", aprovechando además lo ideal del escenario. La idea básica será realizar una decimado de la señal que sale del filtro apareado, y quedarnos con una de cada `samp_per_sym` muestras. Para esto puede usar el bloque `Keep 1 in N`.

  Ahora bien, la muestra con la que se queda el bloque no tiene porqué ser la que usted está buscando. Por ello será necesario agregar un bloque `Delay` con un valor entre 0 y `samp_per_sym` que tendrá que encontrar manualmente.

  Explique como utilizando `QT GUI Constellation Sink` puede encontrar el valor de delay correcto (y encuéntrelo). Utilice el pulso rectangular.

  La figura 5 muestra los primeros dos pasos del receptor implementados en GNU Radio.

  Verificaremos qué sucede con el pulso SRRC.

Tome un largo de filtro bastante grande (por ejemplo, un largo de 51 símbolos). ¿Qué forma tienen los complejos que salen del sub-sistema de muestreo? Haga la misma prueba pero con un largo de filtro mucho más pequeño (por ejemplo 5 símbolos). ¿Qué sucede ahora con los complejos? ¿A qué se debe este fenómeno? ¿Qué consecuencia tiene usar un filtro de SRRC excesivamente largo?

- **Bloque de Decisión.** Implementar un bloque de decisión para una constelación arbitraria es uno de los aspectos más oscuros dentro del GNURadio-Companion. En este caso, nuestra constelación QPSK es estándar, y es igual a la que asume GNU Radio como QPSK, por lo que no será difícil. En particular, el bloque a utilizar es el `Constellation Decoder`, cuyo único argumento es un `Constellation Object`. Lea la ayuda que trae el bloque y vea porqué el comentario del comienzo del párrafo.

  Para nuestro caso bastará poner la siguiente línea: `digital.constellation_qpsk().base()`. Naturalmente, será necesario importar el módulo `digital` (i.e. agregar un bloque `Import` con la línea `from gnuradio import digital`). También es posible usar el bloque `Constellation Object`, pero su documentación es inexistente, por lo que se prefirió el método anterior.

  Edite el archivo `constellacion.cc` (que se encuentra, en la versión 3.8, en el directorio `gr-digital/lib/`) y busque el método `constellation_qpsk()`. Verifique que la constelación asumida para QPSK por los desarrolladores de GNU Radio es la que utilizamos en este obligatorio.

# 5. Cálculo del SER

Una vez armado el receptor QPSK es hora de verificar su correcto funcionamiento y verificar algunas de las fórmulas teóricas. Lo primero será implementar en el GNU Radio algún mecanismo para estimar la tasa de error de símbolos (Symbol Error Rate, SER). La idea será comparar lo enviado con lo recibido.

Para comparar dos flujos de símbolos como los que tenemos, podemos realizar el siguiente procedimiento: lo primero es convertir el stream de bytes a números (mediante un par de bloques `Char to Float`). Luego habría que restarlos (bloque `Substract`) y si el resultado es distinto de cero significa un error en recepción. Para calcular la probabilidad de error se podría tomar el signo del valor absoluto de esta resta, lo que no es fácil de hacer con los bloques disponibles. Lo que haremos será armar un complejo con parte imaginaria constante igual a cero y parte real la salida de la resta (utilizando los bloques `Float to Complex` y `Constant Source`) y a este complejo le calcularemos el módulo (bloque `Complex to Mag`). Como este valor puede estar entre 0 y 3, falta pasar esto por un bloque `Threshold` con parámetros de `High` y `Low` en 0.5 (o cualquier valor entre 0 y 1). Por último, si queremos el promedio temporal utilizaremos el bloque `Moving Average` con un gran valor de `Length` y un `Scale` igual al inverso de este último.

La figura 6 muestra un método para calcular el SER. El valor del promedio móvil lo podemos consultar mediante un bloque `QT GUI Number Sink`.

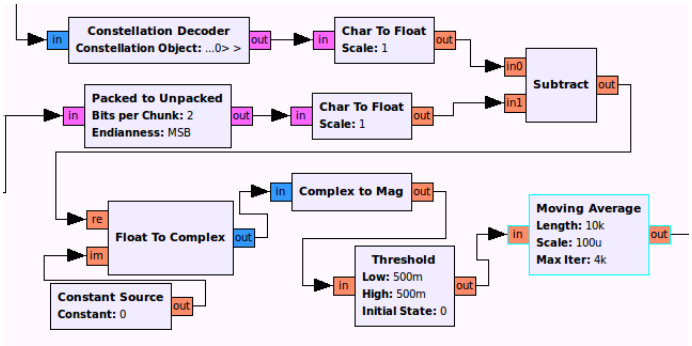


Figura 6: Método para calcular el SER. La salida del `Constellation Decoder` es lo recibido, y la salida del `Packed to Unpacked` es el símbolo transmitido.

Verifique que cualquiera sea su implementación para calcular la tasa de error, le dará positiva y muy cercana a uno si utiliza un pulso SRRC, y cero si utiliza un pulso rectangular. ¿Qué sucedió? Agregue un retardo a la señal de referencia (lo que sale del bloque `Packed to Unpacked` en la figura 2) igual a `2*int(math.floor(len_sym_srrc/2))` y verifique que la tasa de error ahora es cero. Explique porqué fue necesario este retardo y de dónde se obtiene ese valor.

Tomando en cuenta que el valor de `Noise Voltage` en el bloque `Channel Model` es la desviación estándar del ruido gaussiano (i.e. $\sigma$, que no debe confundirse con la varianza $\sigma^2$), verifique la fórmula de la probabilidad de error vista en el teórico para distintos valores de potencia de ruido. Para el cálculo del SNR puede utilizar el bloque `MPSK SNR Estimator`. Interesa en particular ver qué sucede con un pulso de Nyquist perfecto (como el rectangular en este caso) y uno imperfecto (como el de SRRC truncado). Además, puede agregar errores de fase y verificar su efecto poniendo un único `Tap` complejo en el bloque de canal.