Capítulo 7

Integración en GNU Radio

"The radio is playing all the usual, and what's a wonder wall anyway?"
— Fran Healy

7.1. Introducción

Hasta principio de los 2000s, implementar un sistema de RF era una tarea muy costosa debido a la especificidad de los equipos de hardware requeridos y a la necesidad de contar con costosas licencias para el software utilizado en la operación de los mismos. Esto cambió con la aparición de un nuevo paradigma para la construcción de sistemas de RF llamado *Radio Definida por Software* (*SDR*, por sus siglas en inglés). SDR es un sistema de RF donde la mayor cantidad de elementos están implementados en software. Este paradigma tuvo un gran crecimiento durante la última década con la aparición de interfaces de RF para computadoras con costos inferiores a los cientos de dólares, como es el caso del dispositivo RTL-SDR, con un costo de alrededor de U$D 25 [20], como así también debido al crecimiento de la capacidad de procesamientos de los sistemas informáticos y al crecimiento de la comunidad de software libre dedicada al desarrollo de herramientas de SDR, de la cual salieron aplicaciones de gran utilidad que se utilizan tanto en ambientes educativos y aficionados, como así también en la industria [21]. Una de las principales ventajas que tiene SDR sobre las tradicionales implementaciones en hardware dedicado es que permite la actualización y el soporte de múltiples estándares de comunicaciones, siempre y cuando la arquitectura sobre la cual esté instalado este software sea lo suficientemente potente y programable. Esto no es posible en los sistemas tradicionales ya que una actualización o un cambio en una característica requiere el diseño y la implementación de muchos componentes de hardware, ~~lo cual es una~~ tarea costosa y que demanda mucho tiempo [22]. Otra gran ventaja de los sistemas SDR es que reduce enormemente el tiempo de prototipado, ya que no requiere de la fabricación de hardware específico para probar un diseño.

En la Figura 7.1 se muestra un esquema de un sistema SDR, ~~en el cual vemos que está~~ conformado por una etapa de RF encargada de llevar la señal recibida a banda base o de elevar la señal transmitida a una frecuencia de portadora, una etapa de conversión analógica-digital encargada de obtener las muestras digitales de la señal recibida y de hacer el efecto recíproco en la transmisión, y finalmente una computadora que conforma la arquitectura donde correrá el sistema SDR. En un SDR ideal la etapa de RF se implementa dentro del software, sin embargo esto no es simple de realizar debido a que para lograrlo se debe muestrear la señal de entrada a una frecuencia por encima del doble de la portadora, la cual puede estar por encima de los cientos de megahertz o incluso el gigahertz, solo para obtener información de una señal cuyo ancho de banda es, en general, al menos un orden menor que

esta frecuencia.

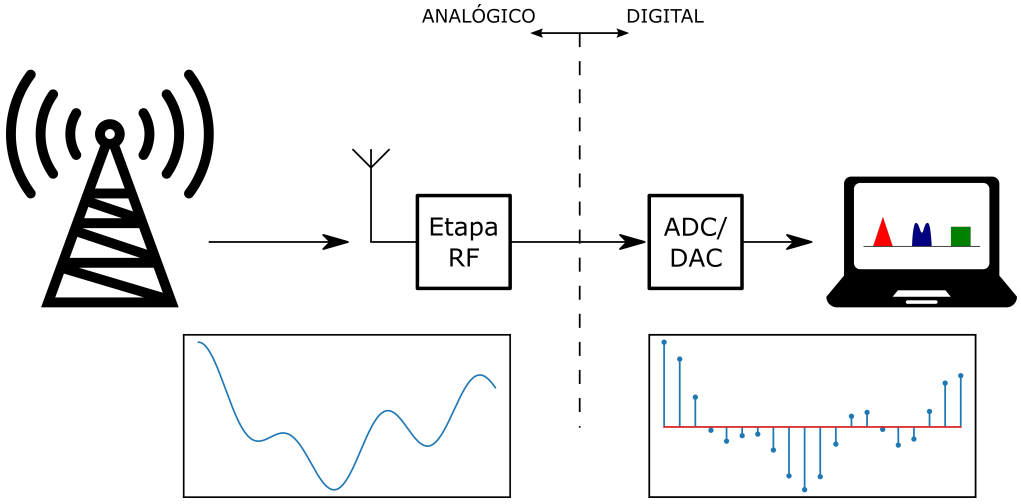


Figura 7.1: Esquema de un sistema de radio definida por software.

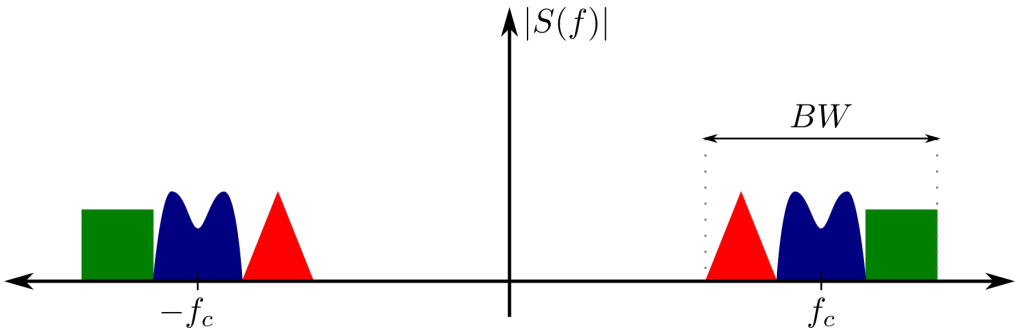
Dentro del marco de aplicaciones de SDR, *GNURadio* es el entorno de desarrollo de procesamiento más popular. Esta herramienta provee de bloques de procesamiento de señal que permiten realizar la implementación de sistemas SDR mediante un entorno gráfico llamado *GNURadio Companion*, el cual es semejante al que proveen herramientas privativas como Simulink® de la empresa MathWorks® o LabVIEW™ de National Instruments™. Aparte de permitir la utilización de las librerías de bloques incluidas en la instalación, GNURadio provee de herramientas e instrucciones para poder generar bloques personalizados mediante programación utilizando los lenguajes `C++` o `Python`, como así también permite compartir las librerías creadas entre usuarios. GNURadio está disponible en los sistemas operativos Microsoft Windows y Linux, sin embargo la versión de Linux es la única que cuenta con soporte oficial. De todas maneras la versión oficial puede ser ejecutada en Windows utilizando el *Windows Subsystem for Linux*, el cual permite ejecutar una instancia de Linux dentro de Windows de una manera más eficiente que ejecutando una máquina virtual.

GNURadio toma un rol vital en este proyecto, proveyendo no solo la capacidad de realizar la implementación de todo el sistema conformador de haz en software mediante la programación de bloques, sino que, además, entrega un entorno de simulación y prueba de diseños que permite verificar el funcionamiento del sistema en tiempo real.

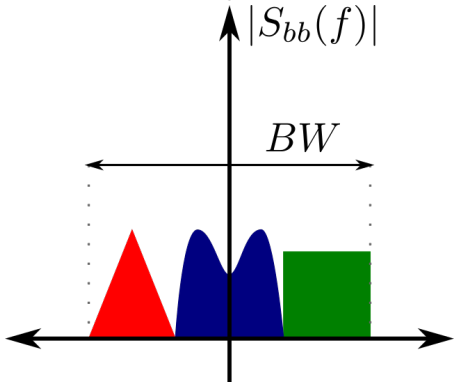
7.2. El modelo en banda base

Debido a que, como se dijo, al trabajar en SDR las señales deben ser llevadas a banda base para disminuir la frecuencia de muestreo requerida en el software, las muestras con las que se trabaja son de tipo complejo, algo que en principio puede resultar poco intuitivo para el usuario. Para demostrar la relación entre estas muestras complejas y las señales originales se analiza, primero, el espectro de señales genéricas de RF montadas sobre una portadora f_c. Si se realiza la transmisión de tres señales reales las cuales se montan sobre una portadora f_c, con un ancho de banda total $BW < f_c/2$, la magnitud de su espectro es simétrica, como se muestra en la Figura 7.2a, y su espectro de fase es antisimétrico [23].

Si a estas tres señales se las lleva de f_c a banda base se obtiene el espectro de la Figura 7.2b, cuya magnitud no es simétrica, y que, por ende, representa a una señal compleja. El espectro de esta señal



(a) Magnitud del espectro de tres señales genéricas montadas sobre una portadora f_c.



(b) Magnitud del espectro en banda base de la señal $S(f)$.

en banda base puede escribirse como un corrimiento del espectro de la señal original $S(f)$ haciendo:

$$S_{bb}(f) = \begin{cases} \sqrt{2}S(f+f_c) & f + fc > 0 \\ 0 & \text{c.o.c} \end{cases}, \tag{7.1}$$

entonces la señal transmitida $S(f)$ puede escribirse a partir de su representación en banda base como [21]:

$$S(f) = \frac{1}{\sqrt{2}}S_{bb}(f - f_c) + \frac{1}{\sqrt{2}}S^*_{bb}(f + f_c) \tag{7.2}$$

La representación de $S(f)$ en el dominio temporal puede obtenerse aplicando la antitransformada de Fourier a la Ecuación 7.2, obteniendo la siguiente expresión:

$$\begin{aligned} s(t) &= \frac{1}{\sqrt{2}}s_{bb}(t)e^{j2\pi f_c t} + \frac{1}{\sqrt{2}}s^*_{bb}(t)e^{-j2\pi f_c t} \\ &= \frac{1}{\sqrt{2}} \cdot 2 \cdot \text{Re}\left\{s_{bb}(t)e^{j2\pi f_c t}\right\} \\ &= \sqrt{2}\text{Re}\{s_{bb}(t)\}\cos(2\pi f_c t) - \sqrt{2}\text{Im}\{s_{bb}(t)\}\sin(2\pi f_c t) \end{aligned} \tag{7.3}$$

siendo $s_{bb}(t)$ la representación en el dominio temporal de $S_{bb}(f)$. La expresión de la Ecuación 7.3 muestra la forma de onda en el receptor antes de realizar la conversión a banda base. Si se desea recuperar la señal $s_{bb}(t)$, es decir, la porción del espectro de la señal $s(t)$ que contiene la información

de interés, puede multiplicarse la Ecuación 7.3 por $\sqrt{2}\cos(2\pi f_c t)$ y por $\sqrt{2}\sin(2\pi f_c t)$ obteniendo:

$$s(t)\cdot\sqrt{2}\cos(2\pi f_c t) = \mathrm{Re}\{s_{bb}(t)\}\underbrace{2\cos^2(2\pi f_c t)}_{1+\cos(2\pi\cdot 2 f_c t)} - \mathrm{Im}\{s_{bb}(t)\}\underbrace{2\sin(2\pi f_c t)\cos(2\pi f_c t)}_{\sin(2\pi\cdot 2 f_c t)} \tag{7.4}$$

$$s(t)\cdot\sqrt{2}\sin(2\pi f_c t) = \mathrm{Re}\{s_{bb}(t)\}\underbrace{2\sin(2\pi f_c t)\cos(2\pi f_c t)}_{\sin(2\pi\cdot 2 f_c t)} - \mathrm{Im}\{s_{bb}(t)\}\underbrace{2\sin^2(2\pi f_c t)}_{1-\cos(2\pi\cdot 2 f_c t)} \tag{7.5}$$

Estas expresiones contienen una componente en banda base de valor $\mathrm{Re}\{s_{bb}(t)\}$ y $\mathrm{Im}\{s_{bb}(t)\}$ sumados a términos centrados espectralmente en $2f_c$, los cuales pueden ser filtrados con un filtro pasa-bajos. A partir de lo obtenido en estas ecuaciones se puede realizar el receptor que se muestra en el diagrama de la Figura 7.3 para así obtener las muestras complejas con las cuales se trabaja en SDR.

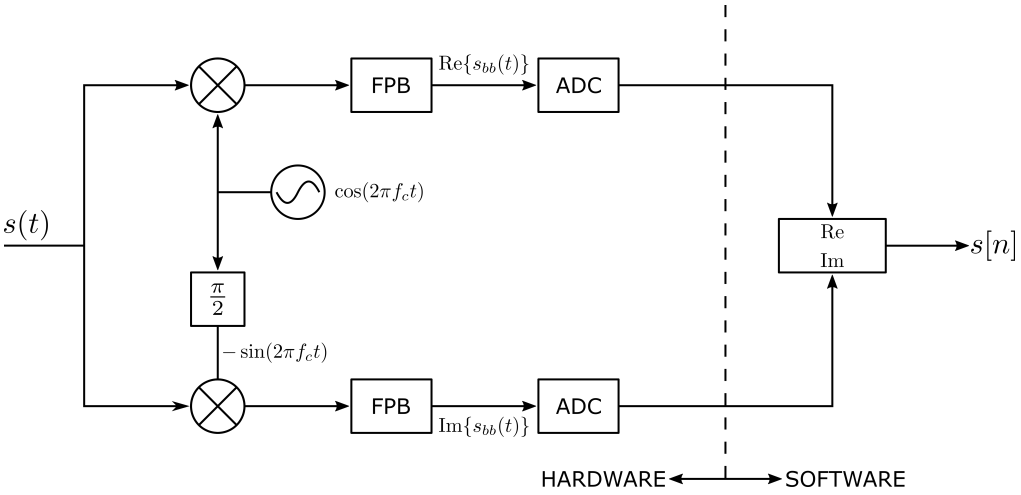


Figura 7.3: Diagrama de bloques de un receptor de un sistema SDR.

7.3. Implementación de módulos

A partir de lo diseñado en el Capítulo 6 se decidió realizar la implementación de todos los módulos utilizando GNURadio para realizar simulaciones en tiempo real del sistema completo y, también, para dejar implementados los subsistemas que posteriormente se instalarán en el PS de la placa de desarrollo. Como ya se dijo, GNURadio permite la creación de bloques utilizando los lenguajes de programación `Python` y `C++`. La implementación en `Python` ofrece las ventajas de ser un lenguaje de alto nivel con gran variedad de librerías de uso libre que permiten la implementación de bloques de una manera muy rápida y sencilla comparada con las implementaciones en `C++`. Sin embargo, al ser un lenguaje intérprete, corre con una gran desventaja en el apartado de rendimiento, ya que las instrucciones de los programas no son ejecutadas directamente por la máquina sino que existe un agente externo, el *intérprete*, que se encarga de leer el código y ejecutarlo. En cambio `C++`, al ser un lenguaje compilado, los programas son convertidos directamente en instrucciones que el procesador puede interpretar, evitando contar con un intermediario y aumentando la eficiencia en la ejecución [24]. Es por esto que en GNURadio solo se recomienda implementar bloques en `Python` en aquellos casos en donde las funciones que realizan no son críticas en el desempeño del sistema, o en aquellos casos en los que se requiere hacer prototipado de bloques para pruebas de funcionamiento rápidas [21]. En esta sección se mencionan algunas características de GNURadio que se utilizaron para la implementación de los bloques necesarios.

7.3.1. Tipos de datos en GNURadio

Las interfaces de los bloques en GNURadio deben tener asignados un tipo de datos definido, los cuales cada uno tiene asignado un color específico. Los tipos de datos permitidos en GNURadio juntos con su código de color se muestran en la Figura 7.4.

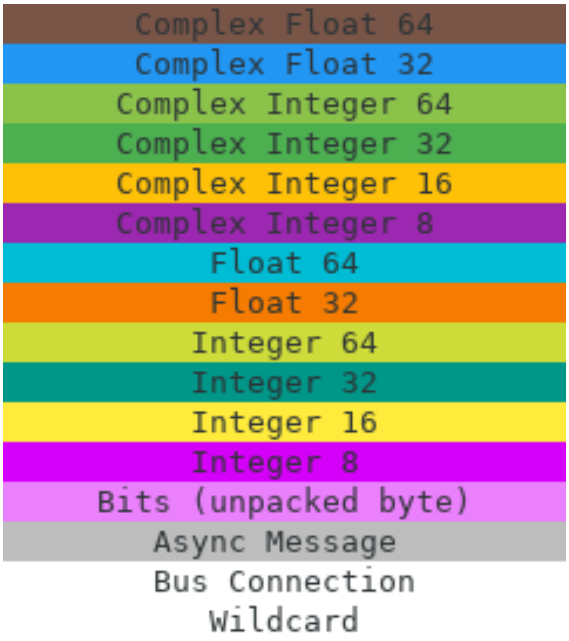


Figura 7.4: Tipos de datos disponibles en GNURadio con su correspondiente identificación de color.

En GNURadio Companion solo pueden conectarse interfaces de bloques que sean del mismo tipo de datos. Para facilitar esta tarea y evitar errores, las interfaces siguen el código de color definido en la Figura 7.4. En la Figura 7.5 se muestra un ejemplo de interconexión correcta e incorrecta entre bloques.

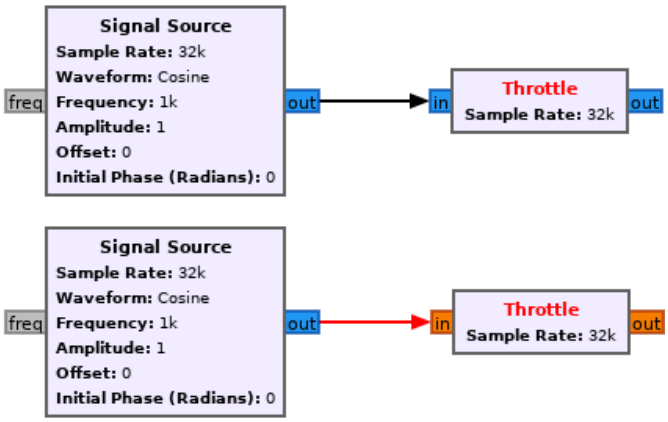


Figura 7.5: Conexiones entre bloques. GNURadio Companion indica con color rojo aquellas conexiones no permitidas.

GNURadio permite, además, el uso de tipo de datos *polimórficos*, los cuales son tipos de datos no explícitos que permiten sobrepasar la exigencia del tipado estricto de `C++` permitiendo declarar un tipo de dato genérico que se define según el contexto de la aplicación. En este proyecto, este tipo de dato

es de gran utilidad a la hora de realizar el envío de los ángulos estimados por el subsistema estimador de DOA al subsistema conformador de haz, ya que, a pesar de que estos datos serán siempre reales, el tamaño de los vectores enviados cambiarán según cuántas sean las señales detectadas.

7.3.2. Callbacks

En las simulaciones que se muestran en la Sección 7.4 es oportuno contar con la posibilidad de cambiar algunos parámetros en tiempo de ejecución, como por ejemplo variar la DOA de una señal simulada para emular la recepción de un satélite LEO, o variar el piso de ruido de manera tal de poder evaluar en el momento cómo una degradación en la señal afecta la estimación de la DOA. El método que permite realizar esto en GNURadio se llama *callback*. Los callbacks son funciones que se declaran en cada bloque para cada variable que se desea poder variar en tiempo de ejecución, y son llamadas por GNURadio Companion en el momento en el que algunas de estas variables se modifica, de manera tal de poder actualizarla en el momento y ver cómo se propaga ese cambio en la simulación.

7.3.3. La librería VOLK

Dentro del proyecto GNURadio existe un subproyecto llamado *Vector-Optimized Library of Kernels* (*VOLK*), el cual consiste en una librería que contiene kernels de código SIMD[1] para la optimización de operaciones matemáticas orientadas a vectores [25]. Estas librerías se encargan de analizar y elegir cuáles son las instrucciones del procesador que optimizan la ejecución de las distintas operaciones matemáticas según en qué arquitectura de hardware se está ejecutando el programa. Utilizando estas librerías se puede alcanzar mejoras de rendimiento de hasta un 40 % [21].

7.3.4. El emulador de ARU.

Debido a que en el momento de realización de este proyecto no se cuenta con el arreglo de antenas ni con el sistema de adquisición implementados, la manera de realizar las pruebas de los sistemas desarrollados es emulando el comportamiento de un ARU. Para esto se implementa el bloque que se muestra en la Figura 7.6.

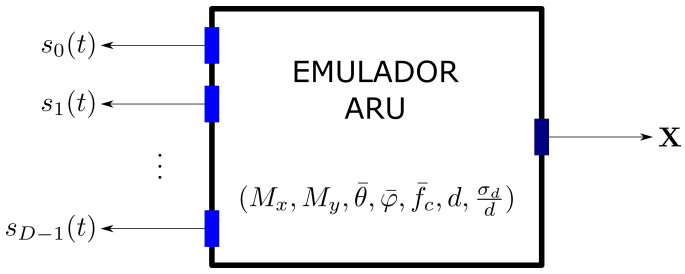


Figura 7.6: Representación como bloque del emulador de ARU para realizar las simulaciones del sistema conformador de haz.

Este bloque recibe D señales en sus entradas y entrega la matriz de salida $\mathbf{X}$ de tamaño $M \times N$, siendo M la cantidad de elementos del arreglo y N la cantidad de muestras temporales. Para hacer esto se basa en el modelo de datos de la Ecuación 3.4 y en la definición del vector de apuntamiento de un ARU definido en la Ecuación 2.13. Este bloque debe ser configurado con la cantidad de elementos del arreglo en ambas direcciones, identificados por M_x y M_y, las direcciones de arribo de las D señales emuladas, identificadas con los símbolos $\bar{\theta}$ y $\bar{\varphi}$, las frecuencias de portadoras de las señales recibidas $\bar{f}_c$, la distancia de separación entre elementos del arreglo d y el error en esta distancia $\frac{\sigma_d}{d}$.

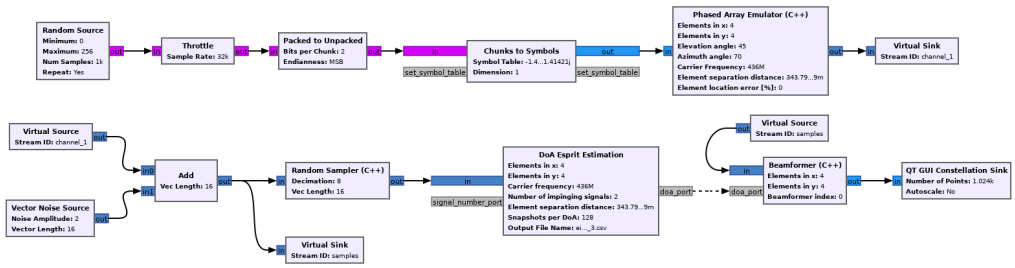
[1] *Single Instruction, Multiple Data*: técnica de software que permite alcanzar paralelismo a nivel de datos.

7.4. Simulaciones

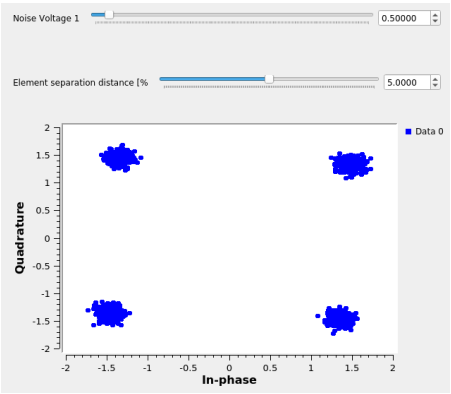
A partir de lo analizado se implementaron los cuatro bloques necesarios para realizar la simulación del sistema completo, estos son:

- **Phased Array Emulator:** emulador de un ARU programado en `C++` utilizando VOLK que recibe muestras de señales complejas y entrega el vector de muestras complejas $\bar{x}$ emulando las salidas de cada elemento de un ARU.
- **Random Sampler:** muestrador aleatorio como se mostró en la Sección 6.3, programado en `C++`, el cual entrega a la salida los vectores de entrada muestreados aleatoriamente.
- **DOA Esprit Estimation:** bloque programado en `Python` que ejecuta el algoritmo de estimación de DOA Esprit, recibiendo vectores de muestras complejas provenientes del muestreador aleatorio y entregando un mensaje polimórfico que contiene vectores reales correspondientes cuyos elementos corresponden a los ángulos de elevación y azimut de las señales detectadas.
- **Beamformer:** bloque programado en `C++` usando VOLK que recibe vectores de muestras complejas y los ángulos de arribo detectados por el bloque DOA Esprit Estimation en forma de mensaje polimórfico y entrega a su salida las muestras de las señales conformadas.

En la Figura 7.7a se muestra una captura del diagrama de bloques de la simulación de una transmisión QPSK utilizando el sistema completo en GNURadio Companion. En la Figura 7.7b se muestra la constelación de la señal entregada por el conformador de haz, la cual corresponde a la señal QPSK transmitida.



(a) Diagrama de bloques del sistema implementado en GNURadio. La zona superior del diagrama corresponde a la transmisión y la parte inferior a la recepción.



(b) Constelación de la salida del sistema conformador de haz al recibir una señal QPSK con una cierta DOA.

Luego de comprobar el correcto funcionamiento del sistema se procede a realizar las pruebas de rendimiento, haciendo hincapié en mediciones de tasa de error de bit (BER) frente a distintos escenarios, y en el procesamiento requerido por cada uno de los subsistemas.

7.4.1. El módulo gr-satellites

El módulo `gr-satellites` desarrollado por Daniel Estévez [26] es uno de los módulos más populares en GNURadio. Este módulo contiene los bloques necesarios para realizar la recepción y decodificación de señales de una gran parte de los satélites de aficionados existentes. Además cuenta con una gran variedad de ejemplos de simulaciones de transmisiones satelitales que pueden ser utilizadas en este proyecto para brindar un grado más de realismo a las pruebas. En particular en este proyecto se utiliza el ejemplo de medición de tasa de error de bit simulando la recepción de información proveniente del satélite LilacSat-1 [27]. Esta simulación realiza esta transmisión iterando para distintos valores de $\frac{E_b}{N_0}$, midiendo la cantidad de errores a la salida y obteniendo una curva de tasa de error de bit. Modificando este ejemplo se agregó el sistema conformador de haz obteniendo el diagrama de bloques que se muestra en la Figura 7.8.

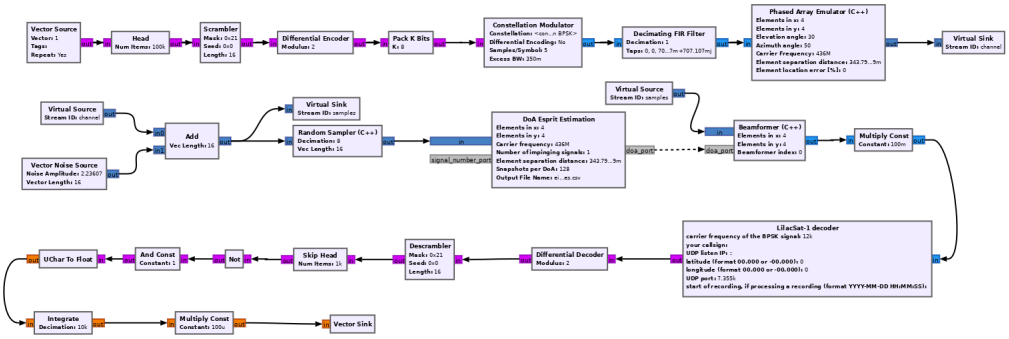


Figura 7.8: Diagrama de bloques utilizado para la medición de tasa de error de bit mediante la simulación de un enlace de comunicación con el satélite LilacSat-1.

7.4.2. Resultados obtenidos

Utilizando el diagrama de la Figura 7.8 se evaluó el desempeño del sistema obteniendo las curvas de tasa de error de bit en función de $\frac{E_b}{N_0}$ para distintos valores de $\frac{\sigma_d}{d}$, obteniendo la gráfica que se muestra en la Figura 7.9.

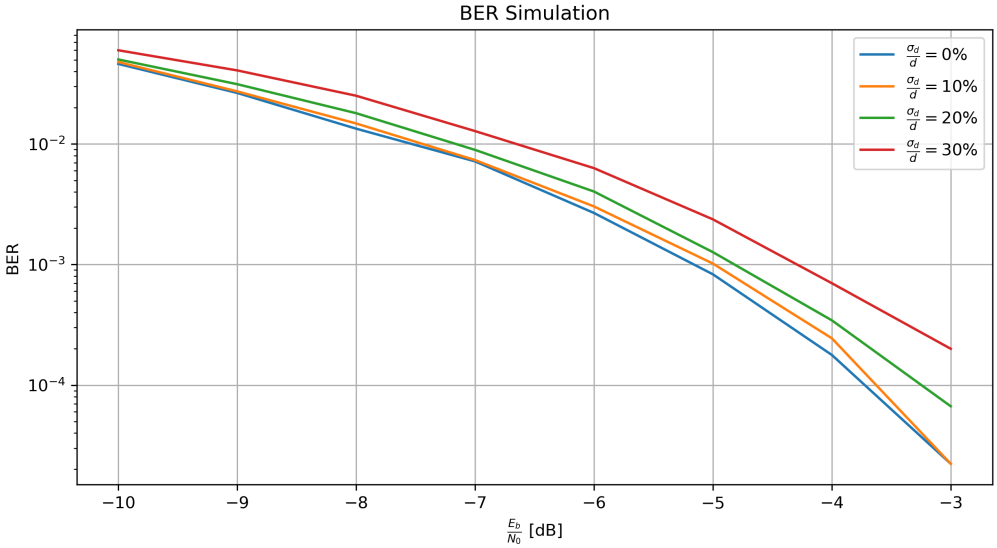


Figura 7.9: Curvas de BER en función de $\frac{E_b}{N_0}$ para distintos valores de errores en la separación de elementos del ARU.

Como puede verse, para un error en la ubicación de elementos del 30 % el desempeño cae 1 dB para este tipo de transmisión.

7.4.3. Requerimientos de procesamiento

Finalmente, se midió el consumo de procesador de cada bloque del sistema conformador de haz durante el tiempo de ejecución utilizando la función `top -H` incluida en Linux, la cual permite ver en tiempo real el consumo de cada proceso que está corriendo en el sistema, obteniéndose los resultados de la Tabla 7.1[2].

Bloque	**CPU [%]**
DOA Esprit Estimation	99,3
Beamformer	3,7
Random Sampler	2,3
Phased Array Emulator	2

Tabla 7.1: Consumo de procesador correspondiente a cada bloque del sistema conformador de haz.

Como puede observarse, el bloque estimador de DOA con ESPRIT es el bloque con peor rendimiento en el sistema, principalmente debido a que es el único implementado en `Python`, por ende, si se desea incrementar el rendimiento para la implementación en la placa de desarrollo, deberá rehacerse en lenguaje `C++`, utilizando la librería `Eigen` [28] para la implementación de la SVD.

[2] El hecho de que la suma de los porcentajes sea superior al 100 % se debe a que la simulación está corriendo en una computadora con un procesador de 8 núcleos, y GNURadio separa la ejecución de cada bloque en un hilo separado para poder ubicar a cada uno en el procesador menos ocupado.