Práctica 1: PROGRAMACIÓN EN RADIO DEFINIDA POR SOFTWARE (GNURADIO)

Juan Sebastián Mendoza García - 2221663
Yeisson Fabian Fernandez Renoga - 2220397
David Leonardo Rodríguez Rosas - 2220413

https://github.com/fernandez7302004-droid/COM_2_A1_G10

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Universidad Industrial de Santander

28 de Febrero de 2028

***Abstract*—In this laboratory practice, user-defined signal processing blocks were implemented and evaluated in the GNU Radio Companion environment. The main objective was to transcend the use of predetermined libraries by algorithmically programming three synchronous components in Python: an accumulator, a discrete differentiator, and a statistical analyzer. By injecting test signals, the logical and mathematical functioning of each block was verified. During development, errors in the base codes were identified and corrected, such as the loss of continuity in the integration of the accumulator between sample batches. The results confirmed the high reliability of the adjusted algorithms compared to theoretical models, validating the ability of Software Defined Radio (SDR) to adapt and customize metric analysis and noise mitigation in real communications systems.**

Palabras clave: *GNU Radio, Procesamiento digital de señales, Python, Radio Definida por Software (SDR), Sistemas síncronos.*

I Introducción

El constante desarrollo tecnológico en el campo de las telecomunicaciones impulsó la evolución hacia la Radio Definida por Software (SDR, por sus siglas en inglés), delegando el procesamiento de las señales a entornos computacionales. En esta práctica se abordó el uso del entorno GNU Radio Companion (GRC) para el procesamiento de señales en tiempo real. La práctica se enfocó en trascender el uso de la interfaz gráfica para adentrarse en la programación de algoritmos propios a nivel de código.

El objetivo principal consistió en desarrollar competencias que permitieran la interacción directa con la programación de funciones mediante Python. Al estructurar bloques a la medida, se facilitó la comprensión del manejo de señales tipo stream y se amplió la capacidad de análisis frente a fenómenos inherentes a los canales de comunicaciones.

II Metodología

El desarrollo del laboratorio se estructuró en dos fases complementarias: la gestión del entorno colaborativo y la codificación algorítmica. Para la primera fase, se empleó el sistema de control de versiones Git, mediante el cual se crearon ramas de desarrollo individuales a partir de una rama principal. Todo el código se administró de manera local y se sincronizó en la nube para consolidar el trabajo grupal.

Para la segunda fase, la manipulación de las señales requirió la creación de bloques síncronos implementados en el lenguaje de programación Python. Dentro del método work(), se extrajeron los vectores de datos utilizando los arreglos unidimensionales input_items y output_items. Se programaron tres bloques principales: un acumulador, un diferenciador y un tercer bloque que resumía características estadísticas de la señal, tales como el promedio, la media cuadrática, el valor RMS, la potencia promedio y la desviación estándar.

A partir de esta estructura, se ensamblaron los diagramas de bloques en la interfaz gráfica y se ejecutaron las simulaciones correspondientes. Las figuras obtenidas a partir de estas simulaciones permitieron comprobar que los valores estadísticos coincidían con la base teórica matemática de las señales inyectadas.[1]

III Resultados

Los resultados obtenidos en la práctica para cada uno de los bloques propuestos se muestran a continuación.

III-A Acumulador

Para evaluar este bloque, se implementó el diagrama de la Figura 1, inyectando una señal cuadrada previamente acondicionada para oscilar simétricamente entre -1 y 1.

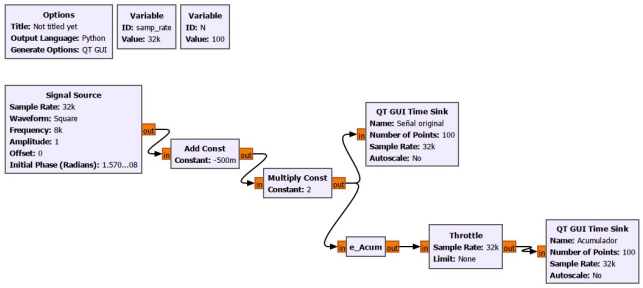

Fig.1 Diagrama de bloques para la evaluación del acumulador.

El código original reiniciaba la suma acumulativa en cada bloque de datos procesado por GNU Radio(debido a que GNU Radio procesa la señal en bloques independientes, ejecutando la función work múltiples veces), generando discontinuidades en la señal de salida. Por ello se añadió una variable que conserva el valor acumulado previo, permitiendo mantener la integración continua entre bloques de muestras.

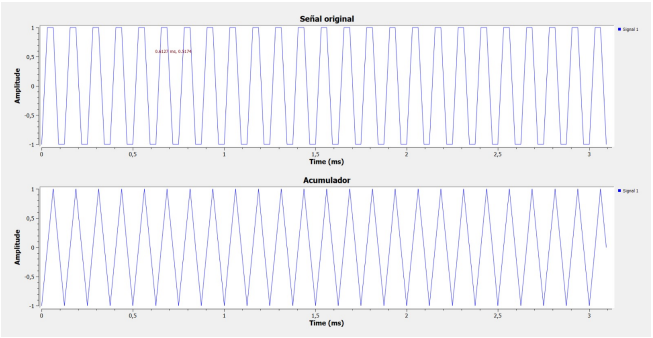

Fig.2 Señal original (cuadrada) y resultante (triangular) tras la acumulación.

Como se evidenció en la Figura 2, el bloque procesó la señal cuadrada de entrada y generó una onda triangular a su salida. Este resultado validó el algoritmo corregido, ya que demostró matemáticamente la integración continua de una señal constante por tramos, manteniendo la pendiente sin caídas abruptas.

III-B Diferenciador

Para evaluar este bloque, se estructuró el esquema de la Figura 3. En este caso, el generador se configuró para emitir una onda triangular, la cual se acondicionó de manera idéntica al montaje anterior para asegurar que su amplitud variara simétricamente entre -1 y 1.

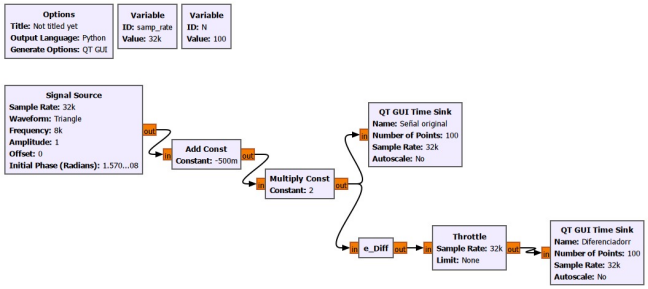

Fig.3 Diagrama de bloques para la evaluación del diferenciador.

Aunque el código original realizaba una resta entre acumulados, el resultado seguía dependiendo de la suma total de muestras anteriores, lo cual corresponde a una integración desplazada y no a una diferenciación. Por esta razón se implementó la diferencia directa entre muestras consecutivas para representar correctamente la derivada discreta de la señal[2]. Al ejecutar el flujo con el bloque corregido, se registraron las gráficas presentadas en la Figura 4.

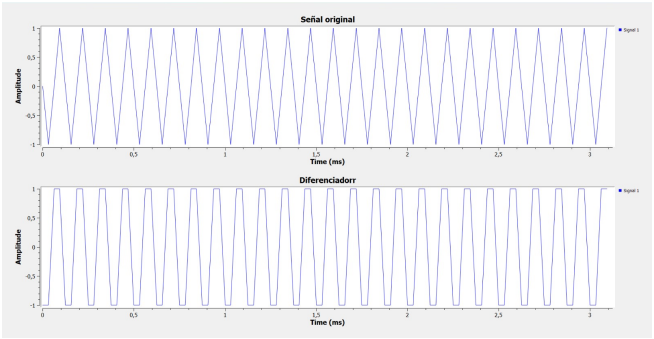

Fig.4 Señal original (triangular) y resultante (cuadrada) tras el proceso de diferenciación.

Como se observó en la Figura 4, al ingresar la onda triangular al bloque e_Diff, el sistema entregó a su salida una onda cuadrada perfectamente sincronizada. Esta gráfica confirmó la exactitud del algoritmo implementado. Matemáticamente, la derivada de una función lineal es una constante, lo que justifica por qué el bloque transformó las rampas de la señal de entrada en los niveles constantes y alternos que conformaron la onda cuadrada de salida.[3]

III-C Analizador Estadístico de Señales

Para la evaluación de este bloque, se estructuró el esquema mostrado en la Figura 5, utilizando un Vector Source para ingresar secuencias numéricas repetitivas y conocidas.

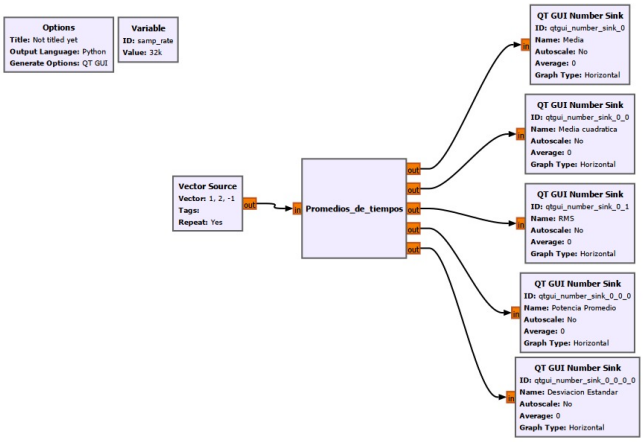

Fig.5 Diagrama de bloques para la evaluación del analizador estadístico de señales.

A diferencia de los bloques anteriores, el código en Python suministrado en la guía base se mantuvo inalterado. [4]

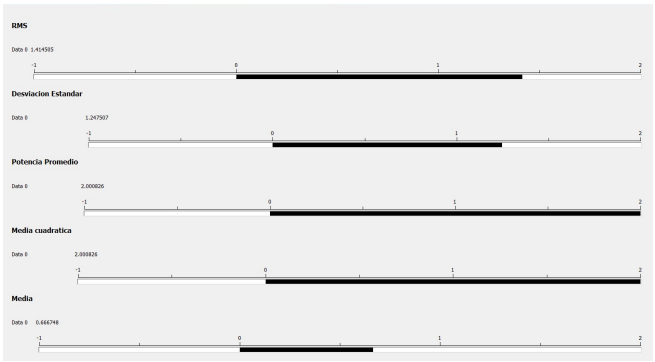

Fig.6 Interfaz gráfica con los valores estadísticos calculados para el vector (1, 2, -1).

La salida de cada parámetro se dirigió a bloques QT GUI Number Sink para su visualización. En la Figura 6 se registró el comportamiento del sistema al ingresar el vector de prueba (1, 2, -1).

Como se evidenció en la Figura 6, los valores procesados por el bloque coincidieron con los cálculos teóricos de la secuencia ingresada. Se observó, por ejemplo, que la potencia promedio y la media cuadrática arrojaron el mismo resultado, lo cual fue matemáticamente congruente con el comportamiento esperado para este tipo de señales.

Para sintetizar la respuesta del sistema de manera eficiente ante distintos escenarios, se introdujeron múltiples secuencias vectoriales, variando tanto el número de muestras como la frecuencia de muestreo. Los datos obtenidos se consolidaron en la Tabla 1.

Vector	Núm. de muestras	Frec. de muestreo [kHz]	RMS	Desviación Estándar	Media	Media cuadrática	Potencia Promedio
(1, 2, -1)	3	32	1.4143	1.2478	0.668	2.000	2.000
(1, 2, -1)	3	4	1.4143	1.2478	0.668	2.000	2.000
(1, -2, 3, -4)	4	32	2.7390	2.692	-0.5	7.503	7.503
(1, -2, 3, -4, 5, -6, 7, -8)	8	32	5.0500	5.0200	-0.500	25.510	25.510
(-1, 2, 1, 0, 2, -2, -1, 1)	8	16	1.4144	1.391	0.250	2.000	2.000
(1, 2, 4, 3, 4, -5)	6	16	3.4390	3.0900	1.500	11.850	11.850
(3, 2, 7, -5)	4	8	4.6630	4.3230	1.750	21.757	21.757

Tabla.1 Parámetros estadísticos obtenidos para diversas secuencias vectoriales.

III-D Aplicaciones

Los bloques de acumulación y diferenciación utilizados en esta práctica tienen aplicaciones en sistemas de comunicaciones digitales y procesamiento de señales muy útiles. Por ejemplo, el diferenciador permite detectar cambios abruptos en una señal, siendo clave en la detección de transiciones binarias y la recuperación de información digital. Por otro lado, el acumulador actúa como un integrador discreto que se puede utilizar en procesos de filtrado y demodulación de señales. Además, el cálculo de parámetros estadísticos, como la media y la varianza, permite analizar características del ruido y la potencia de la señal, aspectos fundamentales en la evaluación del desempeño de sistemas de comunicación. Esto permite que, a nivel industrial, estos bloques se usen en sistemas de monitoreo y control automático; mientras el diferenciador detecta cambios rápidos en variables físicas como temperatura, presión o vibración para el mantenimiento predictivo, el acumulador se emplea en sistemas de medición de energía y procesamiento de señales provenientes de sensores industriales.

IV Conclusiones

- La evaluación visual de las señales procesadas demostró de manera práctica los conceptos teóricos de integración y derivación discreta. Se observó que la transformación de una onda cuadrada a una triangular en el acumulador, así como la conversión de una onda triangular a una cuadrada en el diferenciador.
- A partir de los datos consolidados en la Tabla 1, se concluyó que la variación en la frecuencia de muestreo no alteró el cálculo de parámetros fundamentales como la potencia promedio y el valor RMS. Al someter secuencias idénticas a diferentes tasas de adquisición, los resultados estadísticos se mantuvieron constantes. Este comportamiento demostró la robustez del analizador estadístico implementado frente a distintas velocidades.
- La programación de bloques Out-of-Tree (OOT) mediante Python aportó un valor agregado significativo gracias a su manipulación directa a nivel

de código otorgó un mejora en el control sobre el manejo de las variables y en la precisión de las datos estadísticos extraídos. Esta capacidad nos da mayor flexibilidad para abordar aplicaciones de mayor complejidad,como la estimación de ruido en canales de comunicaciones reales.

References

[1] GNU Radio Project, "Gnu radio," 2024, available: https://www.gnuradio.org.
[2] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 2nd ed. Prentice Hall, 1999.
[3] Wikipedia contributors, "Gnu radio — embedded python block," https://wiki.gnuradio.org/index.php/Embedded_Python_Block, 2026, accessed: 2026-28-02.
[4] Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, *Libro de Comunicaciones Plus Volumen II*. Bucaramanga, Colombia: Universidad Industrial de Santander, 2022, material de apoyo académico.