Laboratorio 1: Programación en Radio Definida por Software (GNU Radio)

Repositorio GitHub: https://github.com/edwinfarid31/2026_1_CommII_A1_G1.git

Edwin Farid Bolaño Perez
Código: 2212264
Universidad Industrial de Santander
Bucaramanga, Colombia
edwin2212262@correo.uis.edu.co

Sergio Andres Cardenas
Código:2222243
Universidad Industrial de Santander
Bucaramanga, Colombia
sergio2222243@correo.uis.edu.co

Johan Sebastian Fandiño Ruiz
Código:2204271
Universidad Industrial de Santander
Bucaramanga, Colombia
Johan2204271@correo.uis.edu.co

***Abstract*—his laboratory report presents the implementation of custom processing blocks in GNU Radio within a software-defined radio framework. A discrete-time accumulator, a differentiator, and a statistical module were developed in Python and integrated into a real-time processing flow. The results highlight the applicability of programmable algorithms for discrete-time signal analysis in noisy environments.**

***Index Terms*—Software-defined radio (SDR), GNU Radio, discrete-time signals, accumulator, differentiator, statistical analysis, real-time processing**

I. INTRODUCCIÓN

La radio definida por software (SDR) ha transformado el diseño de los sistemas de comunicaciones al trasladar funciones tradicionalmente implementadas en hardware hacia algoritmos ejecutados en tiempo real. En este contexto, GNU Radio se consolida como una plataforma flexible para el procesamiento digital de señales, sustentada en una arquitectura modular basada en bloques interconectados.

No obstante, la comprensión profunda de la SDR exige trascender el uso de bloques predefinidos e implementar algoritmos que modelen operaciones fundamentales sobre señales discretas, aprovechando la integración de Python dentro del entorno de desarrollo. En esta práctica se diseñaron bloques personalizados correspondientes a un acumulador, un diferenciador y un módulo de estimación estadística, con el propósito de analizar señales afectadas por ruido y evaluar su desempeño en un sistema de procesamiento en tiempo real.

II. OBJETIVOS

- **Implementación de bloques:** Desarrollo de un acumulador y un diferenciador en Python mediante operaciones vectorizadas para modelar su comportamiento en tiempo discreto.
- **Estimación estadística:** Creación de un módulo para el análisis cuantitativo de señales con ruido.
- **Integración en GNU Radio:** Incorporación de los bloques en un flujo de procesamiento y evaluación de desempeño bajo diversas condiciones.

III. METODOLOGÍA

La implementación se realizó en GNU Radio sobre entorno Windows mediante el desarrollo de bloques personalizados en Python integrados en un flujo de procesamiento en tiempo real. Se programó un bloque acumulador, un bloque diferenciador de primer orden y un módulo de estimación estadística orientado al análisis de señales afectadas por ruido. Los bloques fueron incorporados a una arquitectura modular junto con fuentes de señal y herramientas de visualización, permitiendo evaluar su comportamiento bajo distintas condiciones de entrada y verificar su funcionamiento dentro del sistema de procesamiento digital de señales.

IV. BLOQUE ACUMULADOR

A. Implementación del Bloque Acumulador

El bloque acumulador fue diseñado como un sistema discreto de suma recurrente, descrito por la ecuación:

$$y[n] = y[n-1] + x[n] \tag{1}$$

donde $y[n]$ corresponde a la salida del acumulador en la muestra n, $y[n-1]$ es la salida en la muestra anterior y $x[n]$ representa la señal de entrada. Esta relación define un operador de acumulación equivalente a una integración en tiempo discreto, cuyo comportamiento depende de la memoria del sistema.

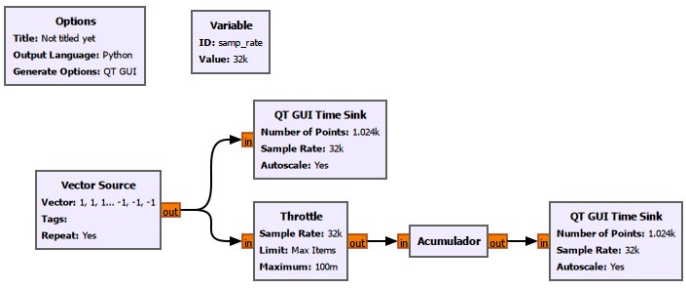

Fig. 1. Bloque acumulador.

Desde el punto de vista funcional, el acumulador transforma una señal de entrada constante en una señal de crecimiento lineal, evidenciando el efecto integrador del sistema. Para validar su comportamiento, se utilizó como señal de prueba el vector $[1, 1, 1, 1, -1, -1, -1, -1]$, el cual genera una forma de onda cuadrada discreta. La señal fue procesada a través de un bloque de control de tasa de muestreo (Throttle) y posteriormente conectada al bloque programable que implementa la ecuación (1). Esta configuración permitió observar la respuesta dinámica del acumulador ante cambios de signo en la señal de entrada.

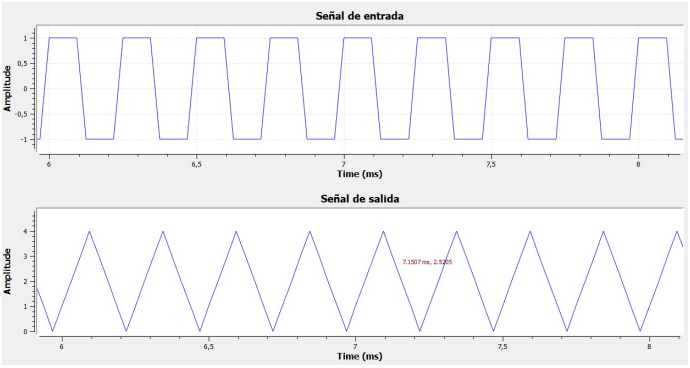

Fig. 2. salida bloque acumulador.

La transformación de una onda cuadrada en una onda triangular confirma que el sistema está operando como un filtro integrador de primer orden, donde la salida es la suma acumulada del área bajo la curva de la señal de entrada.

V. BLOQUE DERIVADOR

Un bloque diferencial calcula la derivada discreta de una señal, es decir, mide el cambio entre una muestra y la anterior. Matemáticamente se expresa como:

$$y[n] = \frac{x[n] - x[n-1]}{T_s} \tag{2}$$

donde:

- $x[n]$ es la muestra actual de la señal.
- $x[n-1]$ es la muestra anterior.
- T_s es el período de muestreo.
- $y[n]$ es la salida diferencial.

Si la señal es constante ($x[n] = x[n-1]$), la salida es cero. Si hay un cambio, la salida refleja la magnitud y dirección de dicho cambio.

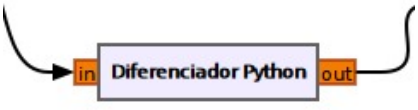

Fig. 3. Bloque derivador.

A. *Configuración del bloque*

Este bloque de código para GNU Radio implementa un proceso síncrono que calcula la variación entre muestras consecutivas de una señal, asegurando que no existan saltos o errores cuando los datos se procesan por ráfagas. Para lograr esto, utiliza una variable de memoria que almacena el último valor del paquete anterior y lo inserta al inicio del nuevo grupo de datos, permitiendo que la función de resta sea continua y fluida. Finalmente, actualiza esta memoria con el dato más reciente para estar listo para el siguiente ciclo, manteniendo así la integridad de la señal en todo momento. Este bloque de Python para GNU Radio implementa un

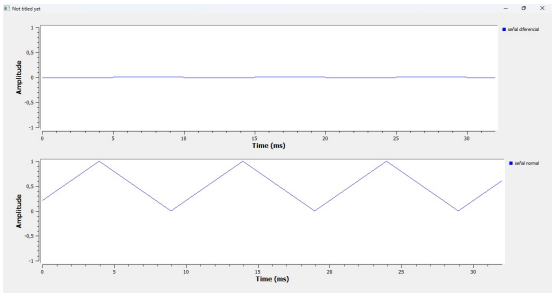

Fig. 4. salida bloque derivador.

bloque síncrono diseñado para procesar flujos de datos tipo `float32` de manera continua. Su lógica principal reside en el método `work`, donde utiliza una variable de estado llamada `self.last_sample` para preservar la continuidad entre ráfagas de datos; esto evita errores de cálculo en los bordes de cada paquete al insertar el último valor del ciclo anterior al inicio del vector actual mediante `np.insert`. Finalmente, emplea la función `np.diff` de NumPy para realizar la resta de elementos adyacentes de forma vectorizada, actualiza la memoria del bloque con la última muestra procesada (`in0[-1]`) y devuelve la longitud del vector de salida para mantener la sincronía del flujo en el programa.

VI. BLOQUE ESTADÍSTICO

Como parte del desarrollo, se implementó un bloque programable orientado al análisis estadístico de señales discretas.

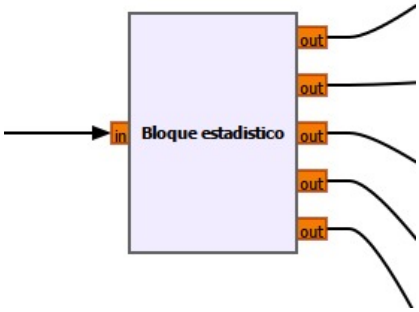

Fig. 5. Bloque estadistico.

A. Configuración del bloque

El bloque estadístico implementado calcula en tiempo real parámetros fundamentales de una señal discreta $x[n]$. A partir del total de muestras procesadas N_{tot}, se obtienen las siguientes magnitudes:

- **Media (promedio):**

$$\bar{x} = \frac{1}{N_{tot}} \sum_{k=1}^{N_{tot}} x[k] \tag{3}$$

- **Media cuadrática:**

$$M_c = \frac{1}{N_{tot}} \sum_{k=1}^{N_{tot}} x^2[k] \tag{4}$$

- **Valor RMS:**

$$RMS = \sqrt{M_c} \tag{5}$$

- **Potencia promedio:**

$$P = (RMS)^2 = M_c \tag{6}$$

- **Desviación estándar:**

$$\sigma = \sqrt{\frac{1}{N_{tot}} \sum_{k=1}^{N_{tot}} (x[k] - \bar{x})^2} \tag{7}$$

Para mantener continuidad en el procesamiento por bloques de datos, el módulo emplea acumuladores que almacenan las sumas parciales y el número total de muestras procesadas.

B. Prueba de validación con señal constante

Con el propósito de verificar la correcta implementación del bloque estadístico, se realizó una prueba utilizando una señal constante de amplitud unitaria $x[n] = 1$. Este caso corresponde a un escenario determinístico cuyas métricas estadísticas pueden determinarse teóricamente, permitiendo validar la coherencia matemática del algoritmo implementado.

Los resultados obtenidos evidencian que la media, la media cuadrática, el valor RMS y la potencia promedio coinciden en valor unitario, mientras que la desviación estándar tiende a cero. Este comportamiento es consistente con la naturaleza constante de la señal, ya que no presenta variaciones respecto a su valor medio. En consecuencia, la prueba confirma el correcto funcionamiento del bloque estadístico bajo condiciones ideales.

TABLE I
RESULTADOS OBTENIDOS PARA UNA SEÑAL CONSTANTE $x[n] = 1$

Parámetro	Valor obtenido
Media	1.000000
Media cuadrática	1.000000
RMS	1.000000
Potencia promedio	1.000000
Desviación estándar	$0.000345 \approx 0$

Observación: La ligera desviación respecto a cero en la estimación de la desviación estándar se atribuye a efectos numéricos asociados a la precisión en punto flotante durante el procesamiento.

VII. Implementación del Sistema de Análisis de Ruido Acústico

Se desarrolló un sistema de análisis estadístico en tiempo real orientado a la caracterización de señales acústicas. El núcleo del sistema corresponde a un bloque personalizado en Python que permite el procesamiento continuo de datos en streaming mediante acumuladores internos.

A. Algoritmo de Procesamiento

Para mantener continuidad entre bloques de datos, el módulo almacena la suma de muestras, la suma de sus cuadrados y el número total de muestras procesadas (N). A partir de estos acumuladores se implementaron las siguientes expresiones:

- **Media:** $\mu = \frac{\sum x_i}{N}$
- **Varianza:** $\sigma^2 = \frac{\sum x_i^2}{N} - \mu^2$
- **Desviación estándar:** $\sigma = \sqrt{\sigma^2}$

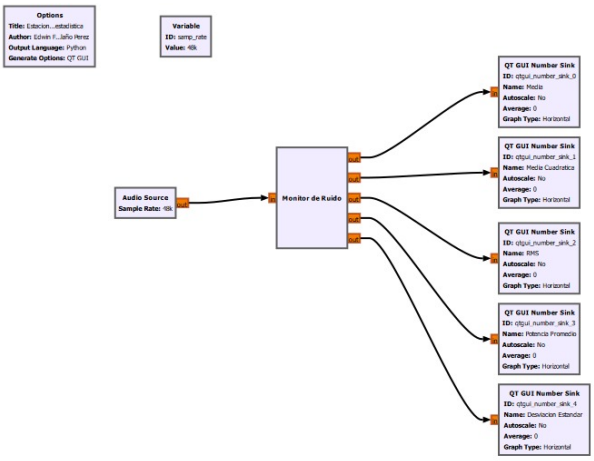

Fig. 6. implementación del sistema de analisis

B. Resultados Experimentales

Al conectar una fuente de audio en tiempo real, se observó que el sistema responde a la presencia de voz humana mediante un incremento significativo en la desviación estándar. Este comportamiento se debe al aumento en la variabilidad de la amplitud de la señal respecto a su valor medio. En condiciones de bajo ruido ambiental se registraron valores de $\sigma \approx 0.001325$, mientras que la media permaneció cercana a cero, indicando una señal centrada y de baja energía promedio.

TABLE II
MÉTRICAS ESTADÍSTICAS EN ESTADO DE REPOSO (SILENCIO)

Métrica	Valor Registrado
Media (y_0)	-0.000000
Media Cuadrática (y_1)	0.000000
RMS (y_2)	0.000326
Potencia Promedio (y_3)	0.000000
Desviación Estándar (y_4)	0.000326

TABLE III
MÉTRICAS ESTADÍSTICAS EN ESTADO ACTIVO (VOZ)

Métrica	**Valor Registrado**
Media (y_0)	-0.000002
Media Cuadrática (y_1)	0.015190
RMS (y_2)	0.123246
Potencia Promedio (y_3)	0.015190
Desviación Estándar (y_4)	0.123246

CONCLUSIONES

- El comportamiento obtenido confirma que la forma de la señal de salida depende directamente de la naturaleza de la señal de entrada. Una entrada de magnitud constante genera una rampa lineal debido al efecto integrador del sistema, mientras que la alternancia de polaridad permite compensar el valor acumulado y producir una señal triangular balanceada. Este resultado evidencia la dependencia del acumulador respecto a su estado previo y valida su correcta implementación como sistema discreto con memoria.
- Los resultados obtenidos en la prueba con señal constante en el bloque estadistico, evidencian la correcta implementación del bloque estadístico, ya que las métricas calculadas coinciden con los valores teóricos esperados para una señal determinística sin variación. La igualdad entre la media, la media cuadrática, el valor RMS y la potencia promedio confirma la coherencia matemática del algoritmo, mientras que la desviación estándar cercana a cero valida la ausencia de dispersión en la señal. En consecuencia, el bloque demuestra un comportamiento consistente y confiable bajo condiciones ideales de entrada.
- La implementación del monitor de ruido en GNU Radio demostró una alta sensibilidad y precisión, logrando que la desviación estándar (y_4) pasara de un valor de reposo de 0.000326 a 0.123246 ante la presencia de voz, lo que valida la capacidad del bloque para detectar variaciones de energía en tiempo real. Gracias al uso de acumuladores y la variable `self.Ntotales`, el sistema mantiene la continuidad estadística procesando cada ráfaga de audio sin perder la historia del evento, mientras que la estabilidad de la media (y_0) cercana a cero confirma la ausencia de errores de desplazamiento DC en la captura. En conclusión, el algoritmo desarrollado es computacionalmente eficiente y robusto para aplicaciones de monitoreo ambiental, permitiendo una visualización clara y profesional de las métricas de potencia y RMS.
- En esta práctica, se exploraron y aplicaron herramientas fundamentales de Radio Definida por Software (SDR) utilizando GNU Radio y bloques personalizados desarrollados en Python. El núcleo del trabajo consistió en el diseño y la evaluación de tres bloques específicos —acumulador, diferenciador y estadístico

REFERENCES

[1] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.

[2] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2007.

[3] GNU Radio Project, "GNU Radio Documentation," [Online]. Available: https://www.gnuradio.org. [Accessed: Feb. 2026].