Práctica 1: Acondicionamiento de Señales y GNU Radio

1er juan david camacho gonzalez
Código: 2210428
Universidad Industrial de Santander
Bucaramanga, Colombia

2do Valentina Arguellez Angulo
Código: 2215670
Universidad Industrial de Santander
Bucaramanga, Colombia

Declaración de Originalidad y Responsabilidad

Los autores de este informe certifican que el contenido aquí presentado es original y ha sido elaborado de manera independiente. Se han utilizado fuentes externas únicamente como referencia y han sido debidamente citadas. Asimismo, los autores asumen plena responsabilidad por la información contenida en este documento.

Uso de IA: Se utilizó el modelo de lenguaje Gemini como asistencia técnica para la estructuración del código en LaTeX, resolución de errores de compilación en Ubuntu/WSL, sintaxis de comandos de Git y revisión gramatical de la justificación técnica. El diseño del filtro, la implementación en GNU Radio y el análisis de resultados fueron desarrollados íntegramente por los autores.

***Resumen*—Este informe muestra el desarrollo e implementación de bloques personalizados para el procesamiento digital de señales en GNU Radio utilizando Python. Se diseñaron módulos como acumuladores y diferenciadores, para analizar el comportamiento de los datos. se manejo la contaminación de señales por ruido Gaussiano mediante un filtro estadístico discreto de media móvil. La eficacia de este acondicionamiento se validó proponiendo su uso en la estabilización de biopotenciales, una etapa crítica en el procesamiento de señales mioeléctricas para sistemas de control de fuerza en la reproducción de movimientos de la mano. Los resultados demuestran que el promediado estadístico atenúa exitosamente las fluctuaciones del ruido, garantizando una señal estable y manteniendo una latencia adecuada para aplicaciones en tiempo real.**

***Index Terms*—GNU Radio, Filtro Media Móvil, Procesamiento de Señales, Python.**

I. Introducción

El procesamiento digital de señales ha experimentado una evolución significativa con la consolidación de la Radio Definida por Software (SDR). Plataformas de código abierto como GNU Radio han transformado el análisis y la transmisión de datos, permitiendo a los desarrolladores trascender el uso de herramientas predefinidas [2], [3] para programar y compilar algoritmos personalizados en lenguajes como Python. Esta flexibilidad es crucial para adaptar el procesamiento de la información a las exigencias matemáticas y físicas de cada entorno.

En el desarrollo de la práctica, se exploran los fundamentos de la creación de bloques de procesamiento en tiempo real. En una primera etapa, se hace la implementación de operaciones matemáticas (acumulación y la diferenciación). Estos módulos conforman la base para el análisis del comportamiento dinámico de los sistemas y la evaluación de tasas de cambio en las señales a lo largo del tiempo.

Por otro lado, un desafío crítico en la instrumentación y adquisición de señales físicas es la presencia de ruido. El acondicionamiento estadístico se vuelve obligatorio en implementaciones de alta precisión; por ejemplo, durante el diseño y validación de un sistema de control de fuerza en la reproducción de movimientos básicos de la mano, donde la presencia de componentes ruidosas en las señales mioeléctricas puede desestabilizar la etapa de inicial. Para soluionar este problema, se propone la implementación de técnicas estadísticas de suavizado.

Finalmente, el documento detalla los resultados obtenidos tras la integración de los módulos de acumulación, diferenciación y filtrado estadístico, en la Sección III se encuentran las conclusiones derivadas del análisis del sistema.

II. Procedimiento

II-A. Acondicionamiento Estadístico: Filtro de Media Móvil

La presencia de ruido Gaussiano es un desafío inherente en la adquisición de señales físicas. Para mitigar las fluctuaciones aleatorias y mejorar la relación señal a ruido (SNR), se diseñó un bloque de procesamiento estadístico personalizado. La estrategia seleccionada corresponde a un filtro discreto de media móvil, regido por la siguiente ecuación de diferencias [1]:

$$y[n] = \frac{1}{N} \sum_{i=0}^{N-1} x[n-i] \tag{1}$$

Donde $N = 10$ representa el tamaño de la ventana de promediado, $x[n]$ es la señal de entrada contaminada y $y[n]$ es la salida estabilizada.

II-A1. Implementación en GNU Radio: La arquitectura del sistema se construyó utilizando el entorno de GNU Radio Companion. El filtro se programó desde cero mediante un *Embedded Python Block*. Como se evidencia en el diagrama de flujo (Fig. 1), la señal original se somete a una fuente de ruido antes de ingresar al bloque diseñado, permitiendo evaluar su desempeño en tiempo real.

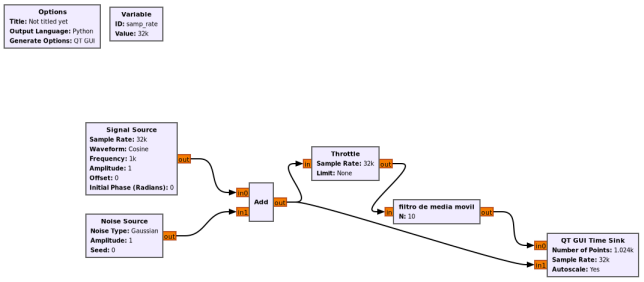

Figura 1. Diagrama de bloques implementado en GNU Radio, destacando el módulo de filtro estadístico en Python.

II-A2. Validación y Aplicación Práctica: La capacidad de programar filtros a medida es un requerimiento crítico en aplicaciones de alta complejidad, como el diseño y validación de sistemas de control de fuerza para la reproducción de movimientos básicos de la mano [4]. En este contexto, las señales mioeléctricas crudas presentan una alta varianza que puede desestabilizar a los actuadores mecánicos. Aplicar un promedio estadístico es el paso previo obligatorio para garantizar un control suave y preciso.

Al ejecutar el sistema, los resultados obtenidos (Fig. 2) demuestran la eficacia del bloque. La estadística aplicada atenúa significativamente los picos del ruido Gaussiano. Aunque persiste un leve rizado debido a una ventana pequeña ($N = 10$), este diseño garantiza una baja latencia computacional, un factor indispensable para el tiempo de respuesta inmediato que exige el control de una prótesis real.

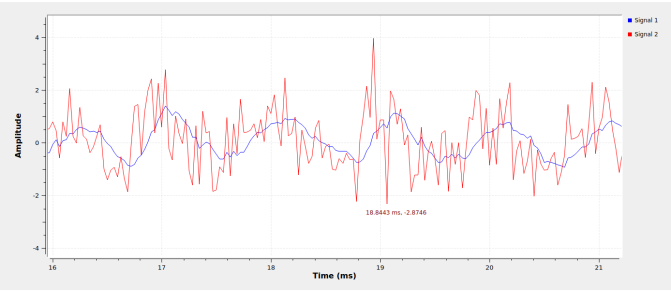

Figura 2. Resultados del procesamiento: Comparación entre la señal contaminada con ruido Gaussiano y la señal estabilizada.

II-B. Acumulador discreto (con memoria)

Un acumulador es un sistema cuya salida corresponde a la suma acumulada de la señal de entrada. En tiempo discreto, si $x[n]$ es la señal de entrada, el acumulador ideal se define como:

$$y[n] = \sum_{k=0}^{n} x[k]. \tag{2}$$

De manera equivalente, puede escribirse en forma recursiva como:

$$y[n] = y[n-1] + x[n], \tag{3}$$

lo cual es especialmente útil para implementación en tiempo real, ya que la salida en el instante n se obtiene sumando la muestra actual a la salida anterior.

II-B1. Motivación de la memoria en GNU Radio (procesamiento por bloques): En GNU Radio, los bloques de tipo `sync_block` procesan la señal en vectores de N muestras durante cada llamada al método `work()`. En consecuencia, si se implementa un acumulador que calcule únicamente la suma acumulada *dentro del vector actual*, se obtiene una salida correcta *solo localmente*, pero el acumulado puede reiniciarse cada vez que `work()` se ejecute de nuevo con un nuevo bloque de datos.

Para reproducir el comportamiento del acumulador ideal $y[n] = y[n-1] + x[n]$ durante una ejecución continua, es necesario almacenar un **estado interno** que represente el valor acumulado al final del bloque anterior. Esta variable de estado (memoria) actúa como condición inicial para el siguiente bloque de muestras.

II-B2. Modelo matemático por bloques: Sea $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$ el vector de entrada en una llamada a `work()` y sea A el acumulado anterior (estado interno). La salida por bloques se define como:

$$\mathbf{y} = A + \mathrm{cumsum}(\mathbf{x}), \tag{4}$$

donde $\mathrm{cumsum}(\cdot)$ representa la suma acumulada elemento a elemento:

$$\mathrm{cumsum}(\mathbf{x}) = [x_0,\ x_0 + x_1,\ \ldots,\ \sum_{i=0}^{N-1} x_i].$$

El estado interno se actualiza con el último valor del acumulado del bloque:

$$A \leftarrow y_{N-1}. \tag{5}$$

De este modo, el acumulador mantiene continuidad temporal: el primer valor del bloque actual parte del acumulado total obtenido en el bloque anterior.

II-B3. Implementación del bloque (Embedded Python Block): La implementación se realizó mediante un bloque personalizado usando *Embedded Python Block* en GNU Radio. El bloque se diseñó como `gr.sync_block` con una entrada y una salida de tipo `float32`. Para conservar continuidad entre bloques se añadió la variable de estado `acum_anterior`, inicializada en 0 y usada como acumulado previo (valor real almacenado como `float`).

II-B3a. Estructura general del algoritmo (descriptiva).: En cada llamada a `work()`, el bloque realiza los siguientes pasos:

- Lectura del vector de entrada: `x = input_items[0]`.
- Preparación del vector de salida: `y0 = output_items[0]`.
- Cálculo acumulado por bloques: `acumulado = acum_anterior + np.cumsum(x)`.
- Actualización de memoria: `acum_anterior = acumulado[N-1]`.

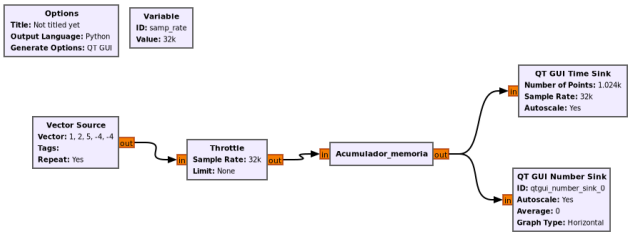

Figura 3. Flujograma de validación del acumulador con memoria. Se observa la salida conectada a *QT GUI Time Sink* y *QT GUI Number Sink*.

Cuadro I
PARÁMETROS GENERALES DEL FLUJOGRAMA DE VALIDACIÓN.

Bloque	Parámetro	Valor
Variable	`samp_rate`	32 kHz
Vector Source	Repeat	Yes
Throttle	Sample Rate	`samp_rate`
QT Time Sink	Sample Rate	`samp_rate`
QT Time Sink	Autoscale	Yes (recomendado)
QT Number Sink	Autoscale	Yes

- Escritura de salida: `y0[:] = acumulado`.
- Retorno de N muestras procesadas: `return len(x)`.

II-B3b. Parámetros y tipos.: El bloque se fijó a:

- **Entrada:** `np.float32`.
- **Salida:** `np.float32`.
- **Estado interno:** `acum_anterior` inicializado en 0.

II-B4. Flujograma de validación (mini comprobación): Para validar el comportamiento del acumulador se construyó un flujo mínimo de prueba en GNU Radio. El objetivo fue observar:

- Correctitud de la suma acumulada (salida coherente con la entrada).
- Continuidad de la salida en ejecución continua (sin reinicios entre bloques).

El diagrama de bloques se muestra en la Fig. 3. Este incluye:

- **Vector Source:** genera la señal de prueba definida por un vector repetitivo.
- **Throttle:** limita la tasa de procesamiento (evita ejecución a máxima velocidad).
- **Acumulador_memoria:** bloque Python implementado.
- **QT GUI Time Sink:** visualización temporal de la salida.
- **QT GUI Number Sink:** monitoreo numérico del acumulado (valor instantáneo).

II-B4a. Parámetros usados en la validación.: Los parámetros generales de la prueba se resumen en la Tabla I. En todos los casos se trabajó con repetición habilitada para observar el comportamiento en ejecución continua.

II-B5. Validación y resultados (tres pruebas complementarias): Se realizaron tres pruebas con diferentes señales de entrada. Cada prueba busca evidenciar un aspecto distinto del acumulador:

- **Prueba 1:** entrada con media distinta de cero $\Rightarrow$ deriva del acumulado (crecimiento).
- **Prueba 2:** entrada de media cero $\Rightarrow$ acumulado acotado (no se va al infinito).
- **Prueba 3:** onda cuadrada bipolar $\Rightarrow$ salida triangular (interpretación como integración discreta).

II-B5a. Prueba 1 (entrada constante, media $\neq 0$).: Se configuró el *Vector Source* con un vector constante:

$$x[n] = 1.$$

En este caso el acumulador ideal produce una rampa:

$$y[n] = 1, 2, 3, 4, \ldots$$

Este resultado verifica la suma acumulada correcta. Además, evidencia la **deriva** inevitable cuando la señal de entrada posee componente DC (media no nula): el acumulado crece aproximadamente de forma lineal con el tiempo.

Recomendación de visualización: como $y[n]$ crece rápidamente, se requiere *autoscale* en el Time Sink o ajustar manualmente el rango vertical para evitar que la señal salga del marco.

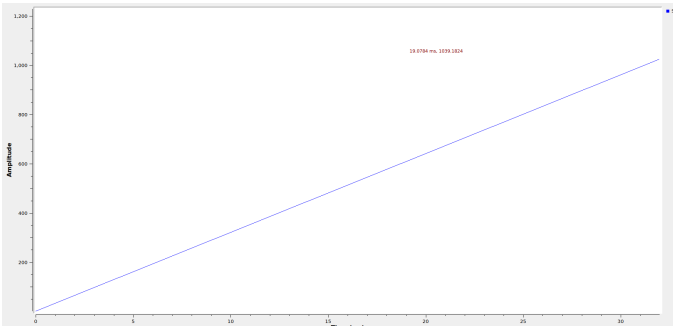

Figura 4. Salida del acumulador con memoria para entrada constante (media $\neq 0$): rampa creciente por deriva del acumulado.

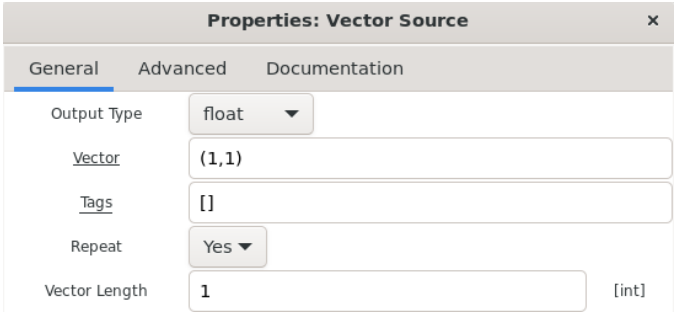

Figura 5. Propiedades del Vector Source en la Prueba 1: vector constante y repetición habilitada.

II-B5b. Prueba 2 (secuencia de media cero: prueba principal).: Con el fin de evitar el crecimiento indefinido del acumulado, se empleó una secuencia repetitiva de **media cero** (suma total por periodo igual a cero):

$$x[n] = [1,\ 2,\ 5,\ -4,\ -4,\ \ldots]$$

La suma por periodo cumple:

$$1 + 2 + 5 - 4 - 4 = 0.$$

En consecuencia, el acumulado **no presenta tendencia** a crecer o decrecer indefinidamente. En su lugar, la salida permanece acotada y refleja las variaciones internas de la suma parcial dentro de cada periodo.

En la Fig. 6 se observa este comportamiento: la señal de salida oscila dentro de un rango y el *Number Sink* permite comprobar que el estado se conserva durante la ejecución continua (no hay reinicios artificiales del acumulado). Esta prueba es especialmente útil porque permite observar el acumulador durante más tiempo sin saturar la escala del *Time Sink*.

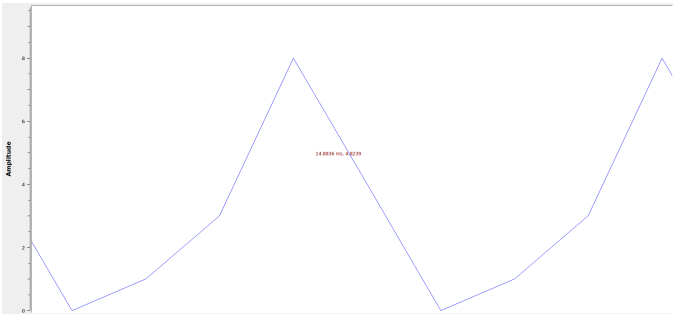

Figura 6. Resultado del acumulador con memoria para una entrada repetitiva de media cero $[1, 2, 5, -4, -4]$: salida acotada y continuidad del estado.

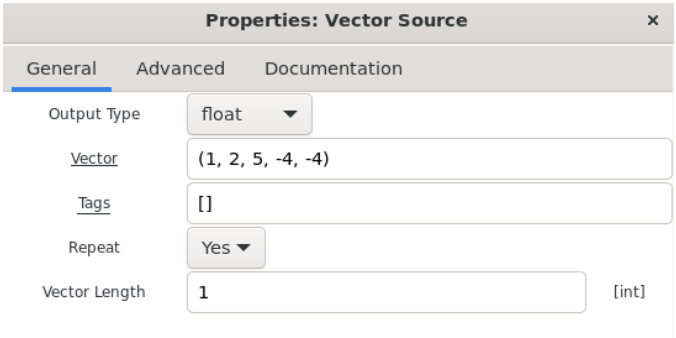

Figura 7. Propiedades del Vector Source en la Prueba 2: vector de media cero y repetición habilitada.

II-B5c. Prueba 3 (onda cuadrada bipolar, media $= 0$).: Para obtener un caso comparable con pruebas típicas de laboratorio (onda cuadrada), se propone una entrada bipolar con valores ± 1 (media cero). Por ejemplo:

$$x[n] = [1, 1, 1, 1, -1, -1, -1, -1, \dots]$$

En este caso, el acumulador incrementa linealmente durante el tramo positivo y decrementa linealmente durante el tramo negativo, generando una forma de onda **triangular** en la salida. Este resultado es consistente con la interpretación del acumulador como una **integración discreta**: integrar una onda cuadrada produce una señal triangular por tramos.

II-B6. Aplicaciones y consideraciones prácticas: El acumulador es una operación base en procesamiento digital de señales. Entre sus aplicaciones típicas se encuentran:

- Aproximación de integrales discretas y cálculo de áreas acumuladas.

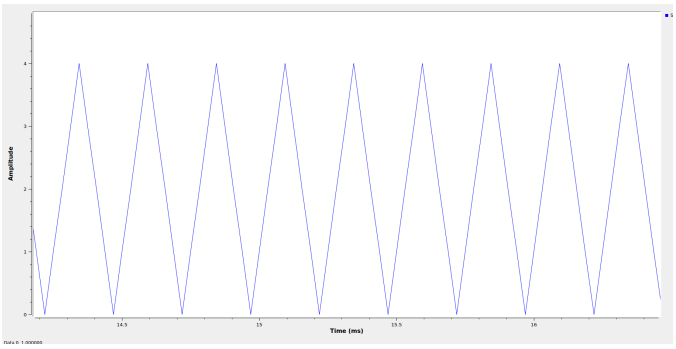

Figura 8. Salida del acumulador con memoria para una entrada cuadrada bipolar (± 1): comportamiento triangular asociado a integración discreta.

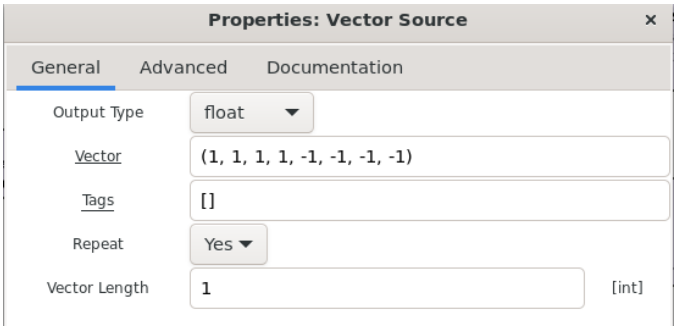

Figura 9. Propiedades del Vector Source en la Prueba 3: secuencia bipolar ± 1 y repetición habilitada.

- Estimación de energía acumulada o sumatorias parciales.
- Construcción de métricas estadísticas acumulativas (promedios, integradores, contadores).

Adicionalmente, los resultados muestran que la visualización debe configurarse adecuadamente: debido al crecimiento potencial del acumulado, es recomendable utilizar *autoscale* o rangos verticales amplios en el *QT GUI Time Sink*. En pruebas de larga duración, las entradas de media cero permiten evaluar estabilidad sin saturación de escala.

II-C. Diferenciador Discreto

Un diferenciador es un sistema cuya salida corresponde a la derivada temporal de la señal de entrada. En tiempo continuo, si la señal de entrada es $x(t)$, el diferenciador ideal está definido como:

$$y(t) = \frac{d}{dt}x(t) \tag{6}$$

Este operador mide la tasa de cambio instantánea de la señal. Por ejemplo, si $x(t) = \sin(\omega t)$, entonces:

$$\frac{d}{dt}\sin(\omega t) = \omega \cos(\omega t) \tag{7}$$

lo cual implica que la derivada de una señal senoidal es otra señal senoidal desfasada $90°$ y con amplitud proporcional a la frecuencia angular. En sistemas discretos, la derivada se aproxima mediante diferencias finitas. La forma más sencilla del diferenciador discreto es:

$$y[n] = x[n] - x[n-1] \tag{8}$$

Esta expresión calcula la variación entre muestras consecutivas, permitiendo estimar la pendiente local de la señal. Si la señal permanece constante, la salida es cero; si la señal presenta un incremento o decremento, la salida toma valores positivos o negativos respectivamente.

II-C1. Implementación en GNU Radio: La implementación del diferenciador se realizó mediante un bloque personalizado en GNU Radio utilizando la opción *Embedded Python Block*. Este bloque fue programado para ejecutar la operación de diferencia discreta entre muestras consecutivas, de acuerdo con la expresión:

$$y[n] = x[n] - x[n-1] \tag{9}$$

El bloque fue diseñado como un `sync_block`, con una entrada y una salida de tipo `float32`, permitiendo procesar la señal muestra a muestra y mantener continuidad entre bloques mediante el almacenamiento del valor previo de la señal. Para la validación del diferenciador se construyó un diagrama de bloques en GNU Radio compuesto por:

- **Signal Source:** Generador de señal, configurado con distintas formas de onda (senoidal, cuadrada) para evaluar el comportamiento del sistema.
- **Throttle:** Bloque utilizado para controlar la tasa de procesamiento y evitar el consumo excesivo de CPU en simulación.
- **Diferenciador (bloque Python):** Encargado de realizar la operación de diferencia discreta.
- **QT GUI Time Sink:** Utilizado para visualizar simultáneamente la señal original y la señal diferenciada en el dominio del tiempo.

La Figura 10 muestra el diagrama de bloques implementado en GNU Radio con la señal senoidal.

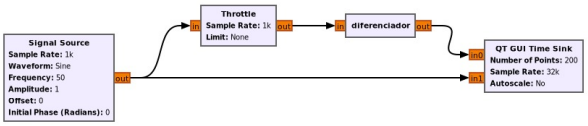

Figura 10. Diagrama de bloques del diferenciador implementado en GNU Radio señal seno.

La Figura 11 muestra el diagrama de bloques implementado en GNU Radio con la señal cuadrada.

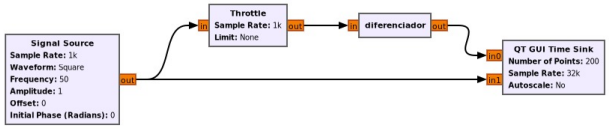

Figura 11. Diagrama de bloques del diferenciador implementado en GNU Radio señal cuadrada.

II-C2. Validación y Aplicación Práctica: El bloque diferenciador implementado fue evaluado utilizando dos tipos de señales de entrada: una señal senoidal y una señal cuadrada. El objetivo fue verificar experimentalmente el comportamiento del sistema frente a variaciones suaves y cambios abruptos en el tiempo, contrastando los resultados obtenidos con el modelo teórico de la derivada.

En el caso de la señal cuadrada, se observó que la salida del diferenciador presentó impulsos de alta amplitud en los instantes donde la señal experimenta transiciones entre sus niveles máximo y mínimo. Durante los intervalos donde la señal permanece constante, la salida fue aproximadamente cero. Este comportamiento es coherente con la teoría, ya que la derivada de una función constante es nula, mientras que en los puntos de cambio brusco la pendiente es elevada. Por lo tanto, el diferenciador actúa como detector de flancos, generando picos positivos en las transiciones ascendentes y picos negativos en las descendentes. La Figura 12 muestra la señal original cuadrada y la señal diferenciada implementada en GNU Radio.

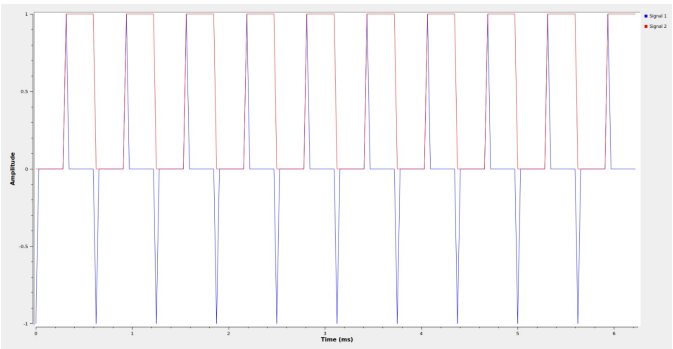

Figura 12. Resultado implementando el dirferenciador para una señal cuadrada.

Por otro lado, al utilizar una señal senoidal como entrada, se obtuvo una señal cosenoidal como salida, lo cual coincide con el resultado matemático esperado:

$$\frac{d}{dt}\sin(\omega t) = \omega\cos(\omega t) \tag{10}$$

Se evidenció además que la señal diferenciada presenta un desfase de $\pi/2$ radianes respecto a la señal original, lo cual es característico del proceso de derivación en funciones trigonométricas. Asimismo, la amplitud de la señal resultante depende de la frecuencia angular ω y de la frecuencia de muestreo utilizada en la implementación discreta.

La Figura 13 muestra la señal original seno y la señal diferenciada implementada en GNU Radio.

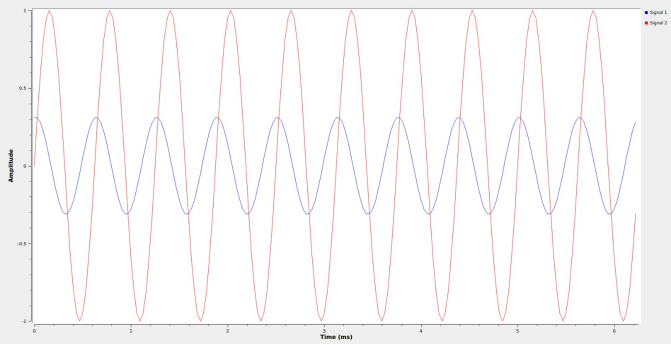

Figura 13. Resultado implementando el dirferenciador para una señal seno.

Es importante mencionar que el diferenciador en tiempo discreto se implementa mediante una aproximación basada en diferencias entre muestras consecutivas. Debido a este proceso de discretización, pueden presentarse pequeñas variaciones en amplitud o ligeras irregularidades cuando la frecuencia de la señal de entrada aumenta o cuando la frecuencia de muestreo no es suficientemente alta.

III. Conclusiones

- El análisis del filtro estadístico de media móvil mostró que la selección del tamaño de la ventana de promediado es muy importante en el diseño. Si bien aumentar la cantidad de muestras atenúa de manera más eficiente la varianza del ruido Gaussiano, esto introduce inevitablemente un retardo temporal en la salida, lo cual representa un parámetro restrictivo en sistemas que operan en tiempo real. A partir de los resultados se concluye que el acumulador implementado reproduce el comportamiento de una **integración discreta** en ejecución continua. Cuando la entrada tiene una componente DC (media distinta de cero), el acumulado presenta **deriva** y crece aproximadamente de forma lineal (Prueba 1). Cuando la entrada tiene **media cero**, el acumulado se mantiene **acotado** y refleja únicamente la estructura de la suma parcial (Prueba 2). Para una entrada cuadrada bipolar (± 1), la salida presenta un comportamiento aproximadamente **triangular**, coherente con la integración discreta por tramos (Prueba 3). Finalmente, la inclusión de la memoria (`acum_anterior`) resulta esencial para garantizar continuidad entre llamadas a `work()` y evitar reinicios del acumulado debido al procesamiento por bloques.
- El acondicionamiento digital de señales mioelectrocas se confirma como una etapa crucial para el diseño y validación de sistemas de control de protesis electromecanicas. La estabilización de estas señales, mediante la reducción de ruido estadístico, es lo que garantiza que los actuadores mecánicos reciban comandos limpios, evitando oscilaciones o respuestas erráticas en la prótesis.
- En general, los resultados experimentales obtenidos confirman el comportamiento teórico de un diferenciador digital. La prueba con señal cuadrada permitió verificar la detección de cambios abruptos, mientras que la prueba con señal senoidal validó la relación matemática entre una función y su derivada. Esto demuestra que el bloque implementado cumple correctamente la función esperada dentro del entorno GNU Radio.

IV. Referencias

Referencias

[1] A. V. Oppenheim y R. W. Schafer, *Tratamiento de señales en tiempo discreto*, 3ra ed. Madrid, España: Pearson Educación, 2011.

[2] A. M. Wyglinski y D. Okin, *Software-Defined Radio for Engineers*. Norwood, MA, USA: Artech House, 2018.

[3] GNU Radio Project, "Embedded Python Block," *GNU Radio Wiki*, 2024. [En línea]. Disponible en: https://wiki.gnuradio.org/index.php/Embedded_Python_Block

[4] M. B. I. Reaz, M. S. Hussain, y F. Mohd-Yasin, "Techniques of EMG signal analysis: detection, processing, classification and applications," *Biological Procedures Online*, vol. 8, pp. 11-35, 2006.