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

ELIANA MARGARITA GONZALEZ APARICIO - 2194677
JOSE EDUARDO BELTRÁN GÓMEZ - 2184677
GEOVANI ANDRES ARENAS CUEVAS - 2170811

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

SEPTIEMBRE 2024

Resumen

Custom blocks such as an accumulator, a differentiator, and a time-averaging block were implemented during the GNU Radio lab practice with the aim of deepening the analysis of digital signals. The addition of a noise generator allowed observing its impact on measurements, highlighting the importance of customization in signal processing and the significant impact of noise on digital measurements. This experience reaffirmed the importance of signal analysis and established a solid foundation for future research in the field of digital communications.

Palabras clave: Potencia promedio , diferenciador, ruido

1. Introducción

El proyecto de GNU Radio permite el desarrollo de bloques de procesamiento de señales que pueden ser escritos tanto en lenguaje de programación Python o C++. Este tipo de módulos son conocidos como out-of-tree, ya que, aunque los módulos serán integrados dentro del catálogo de bloques de GNU Radio no se van a integrar al proyecto para su distribución. La utilidad de los bloques programables en Python disponibles en GNU Radio realizamos dos bloques básicos. El primero que se realizó fue un bloque acumulador, que básicamente entrega la aproximación discreta de una integral de la señal de entrada.

2. Procedimiento

1. Se creó una rama para el desarrollo de la práctica desde el repositorio del navegador a partir del tutoría indicado en la práctica, creando una nueva rama en ese directorio asignándolo para el código de GNURadio y el informe, para de esta manera permitir el trabajo colaborativo.

2. Utilizando el libro guía en la sección 1.2.0.1 se implemento en el bloque de Python el algoritmo sumador y diferenciador.

3. Se implementó un bloque para mostrar la estadística vista en clase y se sugiere una aplicación de los conceptos vistos en clase y se recreó una vista de los resultados.

4. Se utilizó el comando "git status"para verificar los cambios realizados de manera local en el PC. Se utilizó el comando "git add ."para agregarlos al staging area.

3. Resultados

3.1. Creación de bloques programables en GNU Radio

Como una introducción a la utilidad de los bloques programables en Python disponibles en GNU Radio se realizaron dos bloques básicos. El primero que se realizó fue un bloque acumulador, que básicamente entrega la

aproximación discreta de una integral de la señal de entrada. El código de este se puede encontrar en la figura 1.

En segundo lugar, realizamos un bloque diferenciador, que sería entonces el análogo a la derivación continua de una señal discreta. El código de este lo encontramos en la figura 2.

```
import numpy as np
from gnuradio import gr


class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self):  # only default arguments here
        gr.sync_block.__init__(
            self,
            name='2170811_accum',   # will show up in GRC
            in_sig=[np.float32],
            out_sig=[np.float32]
        )

    def work(self, input_items, output_items):
        x = input_items[0]
        y = output_items[0]

        y[:] = np.cumsum(x)

        return len(y)
```

Fig. 1: Código bloque acumulador.

```
import numpy as np
from gnuradio import gr

class DiffAccumulator(gr.sync_block):
    """
    Este bloque calcula la diferencia acumulada de una señal.
    """

    def __init__(self):
        gr.sync_block.__init__(
            self,
            name='Diff_2170811',
            in_sig=[np.float32],
            out_sig=[np.float32]
        )
        self.previous_accumulated_value = 0

    def work(self, input_items, output_items):
        input_signal = input_items[0]
        output_signal = output_items[0]
        num_samples = len(input_signal)

        accumulated_diff = np.cumsum(input_signal) - self.previous_accumulated_value
        self.previous_accumulated_value = accumulated_diff[num_samples - 1]

        output_signal[:] = accumulated_diff
        return len(output_signal)
```

Fig. 2: Código bloque diferenciador.

Por último realizamos un bloque que calcula diferentes promedios de tiempo, en este caso calcula el promedio, la media cuadrada, el valor RMS, la potencia promedio y finalmente la desviación estándar de la señal. Este bloque se usará en un flujograma por sí solo. El código del bloque lo encontramos en la figura 3.

```
import numpy as np
from gnuradio import gr
class blk (gr. sync_block ):
  def __init__ ( self ) : # only default arguments here
    gr. sync_block . __init__ (
    self ,
    name ="2170811_prom", # will show up in GRC
    in_sig =[ np. float32 ],
    out_sig =[ np. float32 ,np. float32 ,np. float32 ,np. float32 ,np. float32 ]
    )
    self . acum_anterior = 0
    self . Ntotales = 0
    self . acum_anterior1 = 0
    self . acum_anterior2 = 0
  def work ( self , input_items , output_items ):
    x = input_items [0] # Señal de entrada .
    y0 = output_items [0] # Promedio de la senial
    y1 = output_items [1] # Media de la senial
    y2 = output_items [2] # RMS de la senial
    y3 = output_items [3] # Potencia promedio de la senial
    y4 = output_items [4] # Desviacion estandar de la senial
    # Calculo del promedio
    N = len (x)
    self . Ntotales = self . Ntotales + N
    acumulado = self . acum_anterior + np. cumsum (x)
    self . acum_anterior = acumulado [N -1]
    y0 [:]= acumulado / self . Ntotales
    # Calculo de la media cuadratica
    x2=np. multiply (x,x)
    acumulado1 = self . acum_anterior1 + np. cumsum (x2)
    self . acum_anterior1 = acumulado [N -1]
    y1 [:] = acumulado1 / self . Ntotales
    # Calculo de la RMS
    y2 [:] = np. sqrt (y1)
    # Calculo de la potencia promedio
    y3 [:] = np. multiply (y2 ,y2)
    # Calculo de la desviacion estandar
    x3 = np. multiply (x-y0 ,x-y0)
    acumulado2 = self . acum_anterior2 + np. cumsum (x3)
    self . acum_anterior2 = acumulado2 [N -1]
    y4 [:] = np. sqrt ( acumulado2 / self . Ntotales )
    return len (x)
```

Fig. 3: Código bloque promedios.

3.2. Creación flujogramas de prueba

Usando estos dos bloques realizamos un montaje simple con un generador de señales continuas, los dos bloques mencionados además de dos ventanas de tiempo. Todo esto para confirmar el correcto funcionamiento de los bloques realizados. Montaje en la figura **??**.

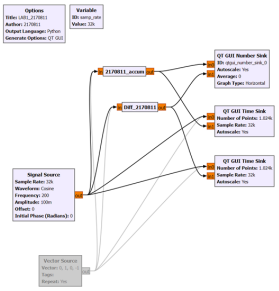

Fig. 4: Montaje prueba bloques Python. Realizado en GNU Radio.

Para el uso del bloque de promedios de tiempo realizamos dos montajes con diferentes bloques para señal de entrada. En el primero, figura 5, usamos un bloque que entrega un vector que nosotros definimos, el resultado de este montaje se usará como referencia para comparar con el siguiente que se realizó.

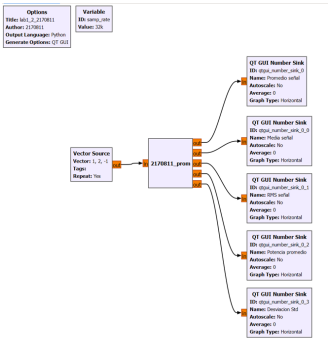

Fig. 5: Montaje de prueba bloque promedios de tiempo con vector de entrada. Realizado en GNU Radio.

Para el segundo, figura 6, usamos dos bloques generadores junto a un sumador, para obtener una señal sinusoidal con ruido. Así entonces podremos observar la diferencia que causa el ruido en las medidas realizadas por nuestro bloque programado en Python en comparación al primer montaje visto en la figura 5.

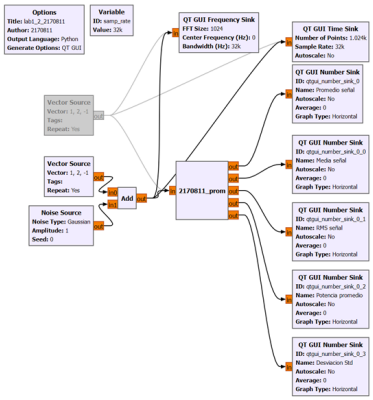

Fig. 6: Montaje prueba bloque promedios de tiempo con ruido. Realizado en GNU Radio.

4. Análisis de resultados

4.1. Bloque sumador y diferenciador

Para este primer montaje de prueba se usó una señal tipo sinusoidal de 200 [Hz] con amplitud de 100 [mV] como señal de entrada, podemos observar los resultados de este montaje en la figura 7 y 8.

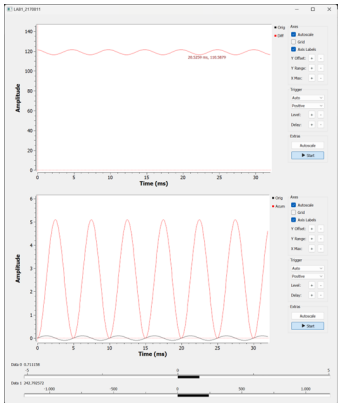

Fig. 7: Ventana de visualización señal de salida. Realizado en GNU Radio.

Se puede observar en la figura 7 el correcto funcionamiento de ambos bloques, en la superior el diferenciador y en la inferior el acumulador. Podemos observar que el bloque diferenciador genera una señal sinusoidal de pequeña amplitud, pero con un componente contante que la desplaza hacia arriba dejando su centro en 120. Observando nuevamente la ventana superior en la figura 8 podemos observar que la misma señal de salida posee saltos entre -150 y 150, este comportamiento nos muestra la manera en la que GNU Radio maneja el procesamiento de señales, este realiza el procesamiento de una señal continua en chunks o bloques, lo cual permite un manejo más eficiente del flujo de datos. Así entonces podemos concluir que la señal de salida es correcta, teniendo en cuenta que nuestro bloque debería derivar la señal de entrada, es decir, entregar otra señal sinusoidal desfazada de la original.

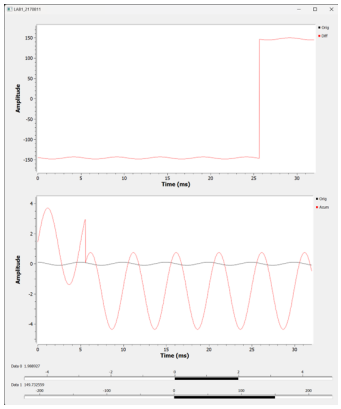

Fig. 8: Ventana de visualización señal de salida cortada. Realizado en GNU Radio.

Ahora observando el comportamiento de la señal se salida tanto en la figura 7 como en la 8 podemos concluir que este bloque también funciona correctamente, entregando la suma continua de nuestra señal de entrada o la integral de esta. Nuevamente en la figura 8 podemos observar los saltos del procesamiento de GNU Radio.

4.2. Bloque promedios en el tiempo

En el primer montaje mostrado en la figura 5, usando la señal vector como entrada obtuvimos los valores de promedio deseados, mostrados en la figura 9. De arriba para abajo entonces podemos observar una ventana de tiempo, donde se puede observar los valores del vector que se repiten. Luego observamos los valores de promedio numéricos los cuales se guardaron en la tabla 1. Por último, tenemos una ventana de visualización en el dominio de la frecuencia la cual nos permitirá observar las frecuencias que componen la señal de entrada.

Ahora, usando la señal de entrada con ruido, se puede observar los valores de salida en la figura 10. En la ventana superior se observa cómo el ruido modifica la señal de entrada de manera aleatoria, esto también se puede observar en la ventana del dominio de la frecuen-

cia, donde la señal de entrada se ve como la señal común que encontraremos en cualquier receptor de radio. En las ventanas de valores numéricos obtenemos valores que quedarán archivados en la tabla 1.

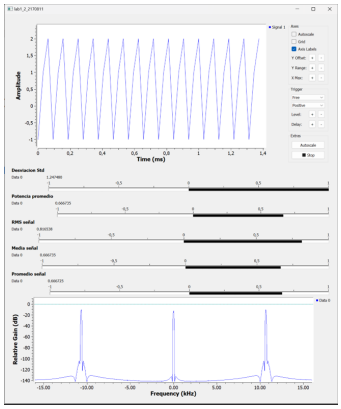

Fig. 9: Ventana de visualización señal de salida sin ruido. Realizado en GNU Radio.

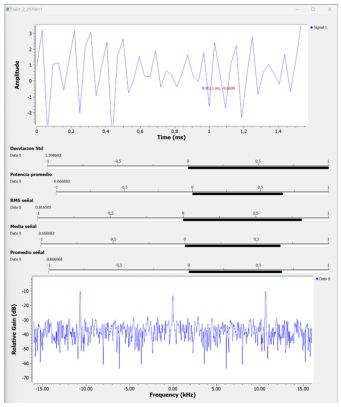

Fig. 10: Ventana de visualización señal de salida con ruido. Realizado en GNU Radio.

Tab. 1: Comparación promedios en el tiempo con y sin ruido en la entrada.

Nombre	Señal sin ruido	Señal con ruido	% Error
Desviación estándar	1.2472	1.5986	28.1751
Potencia promedio	0.6671	0.6667	0.0599
RMS	0.8167	0.8165	0.0245
Media cuadrada	0.6671	0.6667	0.0599
Promedio	0.6671	0.6667	0.0599

Revisando la tabla 1 se pueden hacer dos observaciones, la primera es que el ruido aumenta significativamente la desviación estándar, lo cual es esperado ya que el ruido introduce variaciones aleatorias en la señal de entrada, lo que aumenta la dispersión de los datos. Segundo, la potencia promedio, RMS, media cuadrada, y promedio reaccionaron de la manera esperada al ruido, esto es, siendo robustas al ruido, no mostrando una variación significativa entre los valores originales y los más aproximados a los reales.

5. Conclusiones

- El laboratorio demostró la capacidad de crear bloques personalizados en GNU Radio utilizando Python. Se implementaron con éxito tres bloques: un acumulador (integrador discreto), un diferenciador y un bloque para calcular promedios y estadísticas de señales.
- Se verificó el correcto funcionamiento de los bloques implementados. Mediante flujogramas de prueba y visualización de señales, se comprobó que los bloques se comportan como se esperaba, tanto en el caso ideal (señal sinusoidal pura) como en presencia de ruido.
- Se evidenció el manejo de señales en çhunks.º tramos, por parte de GNU Radio. Los saltos observados en las señales de salida de los bloques diferenciador y acumulador ilustran cómo GNU Radio procesa las señales en bloques para un manejo eficiente del flujo de datos.
- Se analizó el impacto del ruido en las estadísticas de una señal. El experimento con el bloque de promedios demostró que la desviación estándar es más sensible al ruido que otras métricas como el promedio, el RMS y la potencia promedio.
- El laboratorio resaltó la utilidad de los bloques programables en Python para extender las funcionalidades de GNU Radio. La creación de bloques personalizados permite adaptar GNU Radio a necesidades específicas y realizar análisis de señales más complejos

Referencias