Implementación en GnuRadio de un ANC (Adaptive Noise Canceler)

Gonzalo Belcredi (gbelcredi@fing.edu.uy)
Proyecto final del curso Comunicaciones Inalámbricas

Instituto de Ingeniería Eléctica
Facultad de Ingeniería
Universidad de la República
Febrero de 2018

***Resumen*—Este trabajo trata sobre la implementación en GnuRadio de un bloque de procesamiento de señales. Para ello se programó dentro de GnuRadio un Adaptive Noise Canceler (ANC) mediante un estimador secuencial de mínimos cuadrados (LSE).**

I. Introducción

GnuRadio es un Software Libre enfocado a las Radio Definidas por Sofware (SDR), su estructura modular y de código abierto permite crear y ensayar fácilmente bloques de procesamiento de señales.

Un problema común en el área de procesamiento de señales es el filtrado de ruido en un canal. Esto se puede presentar y resolver de distintas maneras dependiendo de la naturaleza del ruido y la capacidad de memoria y procesamiento. En ocasiones nos encontramos con ruidos de los cuáles tenemos cierta información a priori que nos permite implementar técnicas de cancelación.

Por ejemplo a la hora de realizar un electrocardiograma a un feto se busca generalmente aislar el ruido que corresponde al latido de corazón de la madre, una señal que es bien conocida en sus diferentes características pero que naturalmente varía en el tiempo (p.ej las pulsaciones por minuto). Otro ejemplo es la interferencia de 60Hz que tienen muchos componentes electrónicos proveniente del sistema de potencia de AC. En éste ultimo caso, si bien podemos contar con cierta información a priori (sabemos se trata de una sinusoide de frecuencia muy próxima a los 60Hz), desconocemos la amplitud y fase de la misma para cancelarla correctamente.

Para estos casos en ocasiones se diseñan filtros adaptivos que teniendo como entrada una señal de referencia van ajustando su estructura para cancelar al ruido, es el caso del Adaptive Noise Canceler (ANC).

II. Adaptive Noise Canceler

Supongamos que en un canal tenemos la señal $x[n]$ con ruido a cancelar y en un canal secundario tenemos $x_R[n]$ como señal de referencia. Dado que queremos utilizar la señal de referencia para cancelar el ruido, a la primera se la pasa por un filtro cuya salida es restada por la señal $x[n]$, como se muestra en la Figura 1. Se trata luego de ajustar los coeficientes del filtro para minimizar el error cuadrático medio de $\epsilon[n] = x[n] - \hat{x}[n]$.

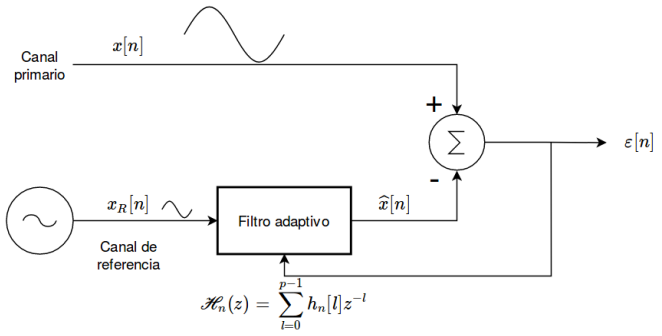

Figura 1. Diagrama de bloques de un Adaptive Noise Canceler [1].

Consideramos una función de costo J definida de la siguiente manera:

$$J[n] = \sum_{k=0}^{n} \epsilon^2[k]$$

$$J[n] = \sum_{k=0}^{n} (x[k] - \hat{x}[k])^2$$

Dado que $\hat{x}$ es la salida de un filtro FIR con coeficientes h_n con p taps, tenemos que:

$$J[n] = \sum_{k=0}^{n} \left(x[k] - \sum_{l=0}^{p-1} h_n[l] x_R[k-l] \right)^2$$

Se puede colocar un factor de olvido λ, $0 \leq \lambda \leq 1$, para darle prioridad a la información más reciente y de esta manera reponder más rápidamente a los cambios en las señales de entrada (ruido o señal de referencia):

$$J[n] = \sum_{k=0}^{n} \lambda^{n-k} \left(x[k] - \sum_{l=0}^{p-1} h_n[l] x_R[k-l] \right)^2$$

Dado que lo que se busca es hallar los coeficientes del filtro para minimizar J esto no cambia si consideramos el siguiente

J':

$$J'[n] = \sum_{k=0}^{n} \frac{1}{\lambda^k} \left(x[k] - \sum_{l=0}^{p-1} h_n[l] x_R[k-l] \right)^2$$

Definimos la estimación de coeficientes del filtro $\hat{\theta}$ como

$$\hat{\theta}[n] = \left[\hat{h}_n[0], \hat{h}_n[1], ..., \hat{h}_n[p-1] \right]^T$$

Sea h la señal de entrada (a no confundir con la respuesta a impulso del filtro):

$$h[n] = [x_R[n], x_R[n-1], ..., x_R[n-p+1]]^T$$

El algoritmo secuencial basado en LSE es resumido por [1] en las siguientes ecuaciones:

$$\hat{\theta}[n] = \hat{\theta}[n-1] + K[n]e[n]$$

Donde,

$$e[n] = x[n] - \sum_{l=0}^{p-1} \hat{h}_{n-1}[l] x_R[n-l]$$

$$K[n] = \frac{\Sigma[n-1]h[n]}{\lambda^n + h^T[n]\Sigma[n-1]h[n]}$$

$$h[n] = [x_R[n], x_R[n-1], ..., x_R[n-p+1]]^T$$

$$\Sigma[n] = \left(\mathcal{I} - K[n]h^T[n] \right) \Sigma[n-1]$$

Observamos que la estimación de los coeficientes del filtro es el valor anterior de los mismos más un factor de corrección proporcional al error $e[n]$. Vemos que si el error es nulo, los coeficientes no cambiarían, pero a medida que ingresa error en el sistema, los coeficientes del filtro se modifican para compensarlo.

III. Implementación de un ANC en GnuRadio

III-A. *Generación de módulos en GnuRadio*

GnuRadio admite la posibilidad de crear módulos Out-Of-Tree (OOT) a través del *gr_modtool*, para ello se debe generar un nuevo módulo (p.ej comina):

```
$gr_modtool newmod comina
```

Para generar un bloque anc_ff que tenga tantas muestras de salida como de muestras entrada se realiza:

```
$gr_modtool add -t sync anc_ff.
```

Estos bloques pueden implementarse tanto en Python como en C++, el *gr_modtool* prepara todos los archivos y métodos necesarios para compilar el módulo en GnuRadio, por lo que para realizar bloques sencillos solo es necesario realizar modificaciones a algunos archivos. En Python por ejemplo luego de crear el bloque anc_ff debemos modificar los archivos:

- *anc_ff.py* en la carpeta python
- *comina_anc_ff.xml* en la carpeta grc

III-B. *Script de python*

La implementación del bloque la debemos realizar en *anc_ff.py*, allí vemos que *gr_modtool* ha implementado la cabecera de varios métodos dentro de la clase principal del bloque:

init Es el constructor del bloque anc, dado que la clase anc hereda de un bloques superiores de gnuradio (*gr_sync_block*) aquí se estipula la cantidad de entradas y salida del bloque así como el tipo de los mismos (enteros, reales, complejos, etc.). También se inicializan aquí los distintos atributos que queramos designar en el bloque (p.ej el factor de olvido λ o p la cantidad de coeficientes del filtro). En nuestro caso construimos el bloque con dos entradas (canal primario y canal de referencia) y dos salidas (estimación del ruido y el error). Todos fueron indicados del tipo float.

work Es donde se implementa el procesamiento de señales. Esta función es invocada periódicamente por GnuRadio y recibe los vectores de entrada los cuales llegan en tamaño variables. Aquí se deben procesar los datos e implementar los vectores de salida. Si queremos conservar estados para la próxima ejecución debemos registrar esta información como atributos de la clase principal a través del método *self*, de lo contrario perderíamos esta información en la próxima ejecución.

Para la implementación en Python se utilizó la librería *Numpy*, lo que facilitó las tareas de algebra lineal involucradas. A este respecto hay que señalar que si bien Python es apropiado para realizar prototipos con poco esfuerzo de programación, la contrapartida es un menor desempeño a nivel de velocidad de procesamiento en ejecución, por ejemplo si lo comparamos con bloques programados enteramente en C++. El código que se puede descargar desde https://github.com/gobelc/anc se muestra en las Figuras 2, 3 y 4.

```
import numpy as np
from gnuradio import gr

class anc_ff(gr.sync_block):
    """
    This block clean-up a signal via minimizing LSE with a reference signal.
    Two outputs available: signal canceled (a tone for example) and the cleaned signal
    (error).
    """
    def __init__(self, forgetting_factor, size, refresh_number):

    def work(self, input_items, output_items):
```

Figura 2. Clase principal.

III-C. *XML para GUI de GnuRadio*

La interfaz gráfica de GnuRadio se conecta al bloque mediante el archivo *comina_anc_ff.xml*, aquí se definen el nombre y etiquetas de los distintos atributos del bloque, así como los tipos de datos.

III-D. *Compilación e instalación del módulo*

Una vez programado el bloque debemos compilar el programa, para ello creamos un directorio build en la raíz del

```
def __init__(self, forgetting_factor, size, refresh_number):
    gr.sync_block.__init__(self,
        name = "anc_ff",
        in_sig = [np.float32,np.float32],
        out_sig = [np.float32,np.float32])
    self.size = size
    self.forgetting_factor = forgetting_factor
    self.refresh_number = refresh_number
    self.sigma_0 = 1
    self.h = (np.ones(self.size)).T
    self.theta = (np.zeros(size)).T
    self.sigma = self.sigma_0*np.identity(size)
    self.K = (np.matmul(self.sigma,self.h)) /
        (1+np.matmul(np.matmul(self.h.T,self.sigma),self.h))
    self.count = 0
```

Figura 3. Función de inicialización.

```
def work(self, input_items, output_items):
    out0 = output_items[0]
    out1 = output_items[1]
    signal = input_items[0][:]
    ref = input_items[1][:]
    salida  =    np.zeros(len(out0))
    error   =   np.ones(len(out1))
    forgetting_factor = self.forgetting_factor
    h = self.h
    theta = self.theta
    sigma = self.sigma
    K = self.K
    size = self.size
    count = self.count
    for n in range(0, len(out0)):
        error[n] = signal[n] - np.matmul(h, theta)
        salida[n] = np.matmul(h, theta)
        h = np.roll(h,1)
        h[0] = ref[n]
        if count>self.refresh_number:
            count = 0
            forgetting_factor = self.forgetting_factor
        denominador = forgetting_factor**n+np.matmul(np.matmul(h.T,sigma),h)
        if denominador != 0:
            K = (np.matmul(sigma,h)) / denominador
        sigma = np.matmul(np.identity(size)-np.matmul(K,(h).T),sigma)
        theta = theta + K*error[n]
        self.h = h
        self.theta = theta
        self.sigma = sigma
        self.K = K
        count = count + 1
    out0[:] = salida
    out1[:] = error
    return len(output_items[0])
```

Figura 4. Implementación de procesamiento de señales.

módulo, y luego ejecutamos:

```
cmake ../
make
make test
sudo make install
sudo ldconfig
```

Si no obtuvimos errores en esta etapa podemos pasar a probar el módulo en el propio GnuRadio.

IV. Resultados

Se realizaron dos pruebas para verificar el funcionamiento del ANC implementado en GnuRadio, en la primera (`test_mag.grc`) (ver Figura 5) se colocó un tono de interferencia sumado a una señal de audio como entrada al canal principal, y luego se agregó al canal de referencia un tono igual al anterior pero con distinta amplitud. Dado que solo se requiere ajustar el módulo de la señal de referencia, el filto debe ser de orden ($p = 1$).

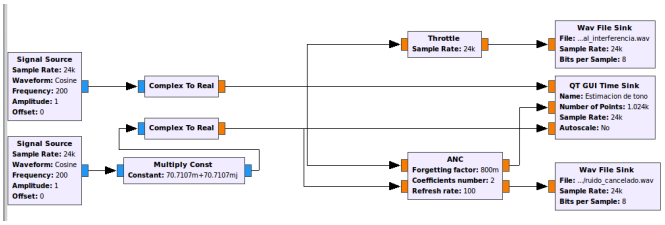

Figura 5. GRC de prueba para la cancelación de tono (amplitud y fase desconocida)

Verificamos que el tono estimado se aproxima al tono de interferencia (ver Figura 7), y como señal de error obtuvimos la señal de audio original.

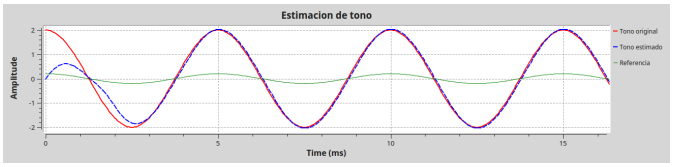

Figura 6. Estimación de magnitud de tono con $p = 1$.

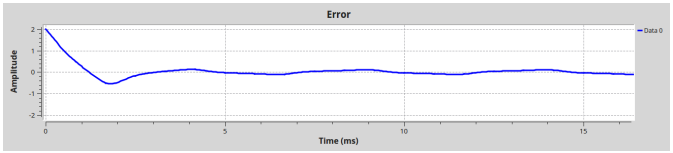

Figura 7. Error de estimación de magnitud de tono con $p = 1$.

Para la segunda prueba (`test_mag_phase.grc`) se desfasó la señal primaria respecto a la señal de referencia, por lo que el ANC debe ajustar tanto el módulo como la fase de la señal de referencia, por ello es necesario colocar un filtro de segundo orden. En este caso fue necesario bajar el factor de olvido para mejorar el funcionamiento del módulo ($\lambda = 0,8$). Los resultados se muestran en las figuras 8 y 9.

La prueba con la mezcla en el canal primario de la señal de interferencia y la señal de audio no produjo resultados tan interesantes como para el caso de la primera prueba, esto lo atribuimos a que el filtro se ajusta no solo a la señal de interferencia sino a toda la señal (la función de costo recordamos era $J[n] = \sum_{k=0}^{n}(x[k] - \hat{x}[k])^2$), y como el filtro ahora ajusta su fase puede seguir con mayor facilidad a la señal de audio ("perdiendo de vista" el tono de interferencia).

V. Conclusiones

En este trabajo se abordaron dos aspectos, por un lado el estudio e implementación de un Adaptive Noise Canceler y por otra parte su integración a GnuRadio. Los resultados obtenidos fueron los esperados aunque se nota el esfuerzo de procesamiento que debe incurrir la computadora al tratarse de bloques programados en Python. El bloque debería ensayarse

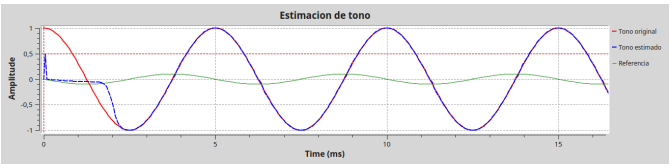

Figura 8. Estimación de magnitud y fase de tono ($\Delta\phi = \pi/2$), $\lambda = 0,8$ y $p = 2$.

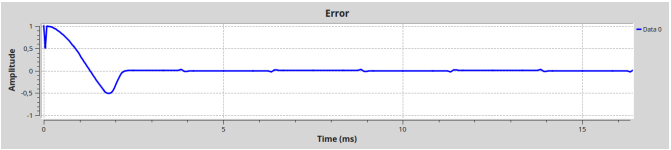

Figura 9. Error en estimación de magnitud y fase de tono ($\Delta\phi = \pi/2$), $\lambda = 0,8$ y $p = 2$.

con otro tipo de señales (además de sinusoidales) para verificar que el filtro adaptivo se desenvuelva correctamente así como eventualmente pasar su implementación a C++.

REFERENCIAS

[1] S.M. Kay. *Fundamentals of Statistical Signal Processing: Detection theory*. Prentice Hall Signal Processing Series. Prentice-Hall PTR, 1998.