EQUIPO DE MEDICIÓN DE ENERGÍA
ETAPA 1

Fernando Fortunati
e-mail: nfortunati@gmail.com

Sebastian Martínez
e-mail: sebajm1363@gmail.com

RESUMEN: *Se aplicarán los conceptos aprendidos en la cátedra Procesamiento Embebido de Señales para el diseño y realización de un medidor de energía.*

PALABRAS CLAVE: Calidad de energía, True RMS, THD, FFT, Ventana Hamming, Python, Tkinter.

1 INTRODUCCIÓN

El objetivo de la 1ra Etapa de este proyecto, consistirá en hacer una primera aproximación al código a implementarse en un microcontrolador, ya que es el objetivo que nos ponemos para la Etapa 2.

En el mismo se abarcaran los conocimientos aprendidos sobre muestreo de señales, aplicación de Ventana y FFT.

Al desear aplicarle la FFT a una señal continua, y que el hecho de hacerlo directamente, hace que se pierda informacion; se le aplicara previamente a la funcion una ventana de Hanning para suvizarla y evitar grandes discontinuidades.

2 ALCANCE

Se propone simular señales de varios tipos con el objetivo de realizar su procesamiento interno para determinar parámetros característicos, tales como el verdadero valor eficaz (True RMS) y la distorsión armónica total (THD).

Para el cálculo de la THD, se aplicará una Transformada Rápida de Fourier (FFT) a la señal previamente muestreada, obteniendo así los componentes espectrales necesarios para la estimación del valor mencionado.

Posteriormente, los valores correspondientes a la señal muestreada, los parámetros utilizados en la Transformada Rápida de Fourier (FFT), así como los resultados del cálculo de True RMS y del índice de Distorsión Armónica Total (THD).

Para ello se diseñara una interfaz en Python, utilizando la librería Tkinter; de forma de poder seleccionar amplitud, frecuencia y tipo de señal (Cuadrada, Senoidal y Triangular).

Luego de generala, la interfaz mostrara graficos de la señal en el tiempo y su espectro en frecuencia. Como asi tambien el valor de la THD y el TRMS.

3 MARCO TEÓRICO

3.1 Muestreo:

El muestreo consiste en convertir una señal continua en una secuencia discreta de valores tomados en instantes periódicos. Según el Teorema de Nyquist, para evitar aliasing, la frecuencia de muestreo f_s debe ser al menos el doble de la máxima frecuencia presente en la señal ($f_s \geq 2f_{max}$).

Para la selección de la frecuencia de muestreo se seguiran los siguientes lineamientos:

- Definir hasta que armónico se deseara mostrar en la FFT.
- Con la frecuencia máxima que se desea ver en el espectro, se seleccionara la frecuencia de muestreo de acuerdo al criterio de Nyquist.
- Se debera adecuar este valor de frecuencia para que sea potencia de 2, de esta forma evitamos el leakage.

EJEMPLO:

Fundamental: 50 Hz

Cantidad de armónicos: 25

Frecuencia Máxima (Fmax): 25 * 50 Hz = 1250 Hz

Frecuencia de Muestreo (Fs): 2 * Fmax = 2500 Hz

Fs corregida: 4096 Hz (exactamente 64 ciclos)

En la práctica, el muestreo no se realiza con impulsos ideales, sino con un tren de pulsos rectangulares, lo que introduce un efecto adicional: la señal muestreada se convoluciona con una función sinc en frecuencia, debido al ancho finito de los pulsos.

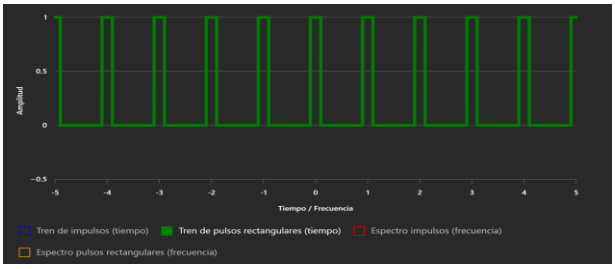

Figura 1: Tren de pulsos rectangulares

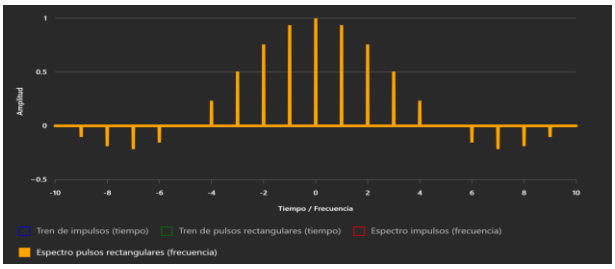

Figura 2: Respuesta en Frecuencia

3.2 Ventana de Hanning:

La **Ventana de Hanning** (o Hann) es una función utilizada en procesamiento digital de señales para reducir el **efecto de fuga espectral** cuando se aplica la Transformada de Fourier a señales finitas. En lugar de cortar la señal abruptamente, se multiplica por una ventana que suaviza los extremos, disminuyendo las discontinuidades.

Su fórmula es:

$$\omega[n] = 0{,}5 * \left(1 - cos\left(\frac{2\pi n}{N-1}\right)\right), 0 \leq n \leq N-1$$

Características:

- Reduce la fuga espectral mejor que una ventana rectangular.
- Atenúa los extremos de la señal a cero.
- Se usa en análisis espectral y filtrado.

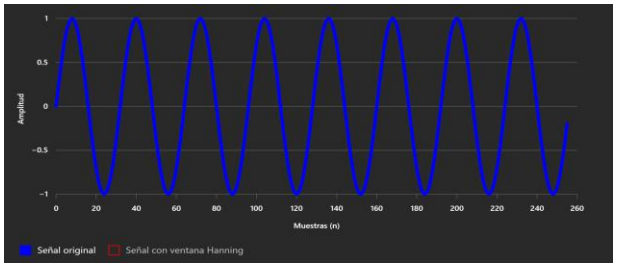

Figura 3: Señal Senoidal (Sin aplicarle la ventana)

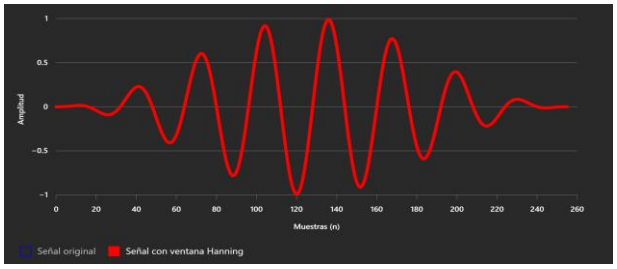

Figura 4: Señal Senoidal (Ventana Aplicada)

Para mas claridad se analizan ahora, los espectros antes y después de aplicarle la ventana:

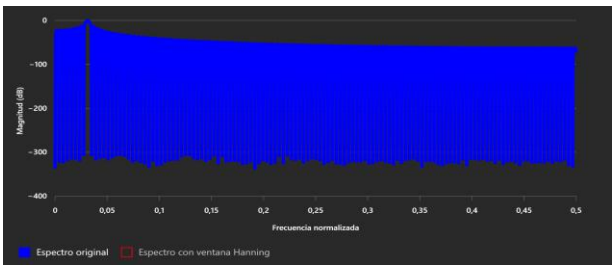

Figura 5: Espectro de la Señal Senoidal (sin ventana)

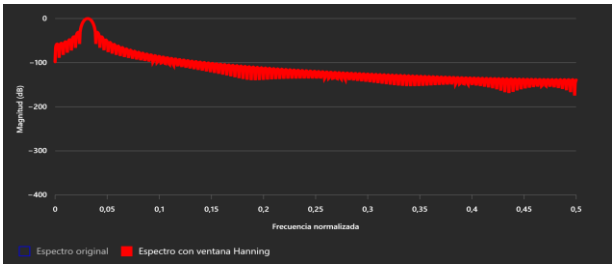

Figura 6: Espectro de la Señal Senoidal (Con ventana)

Como se puede apreciar, cuando no se aplica la ventana hay un alto nivel de componentes de alta frecuenia que no pertencen a la señal original. Aunque el pico principal está en la frecuencia correcta, la energía se dispersa en frecuencias adyacentes, dificultando la interpretación.

En cambio, luego de aplicar la ventana, se aprecia una atenuacion de las componentes de alta frecuencia. Esto se debe a que se suavizaron los extremos de la señal (Figura 4), lo que disminuye la Fuga Espectral. El resultado es un espectro más limpio, con menos energía fuera de la frecuencia principal.

3.3 Transformada Rápida Fourier (FFT):

La Transformada Rápida de Fourier (FFT) es un algoritmo eficiente para calcular la Transformada Discreta de Fourier (DFT), que convierte una señal del dominio del tiempo al dominio de la frecuencia. Esto permite analizar qué componentes frecuenciales forman la señal.

La DFT se define como:

$$X[k] = \sum_{n=0}^{N-1} x[n] * e^{-j2\pi kn/N} \ , k = 0, 1, 2, \dots, N-1$$

Diferencias:

- La DFT directa requiere $O(N^2)$operaciones para Nmuestras.
- La FFT reduce esto a $O(N\log\ N)$, lo que permite análisis rápido incluso para señales largas.
- El punto anterior, en el caso de sistemas embebidos. Mejora la velocidad de procesamiento y la utilizacion de recursos.

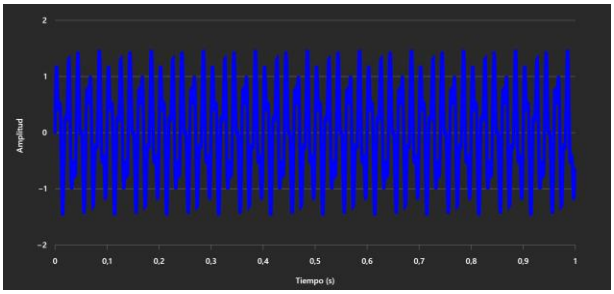

Figura 7: Señales senoidales de 50 y 120 Hz

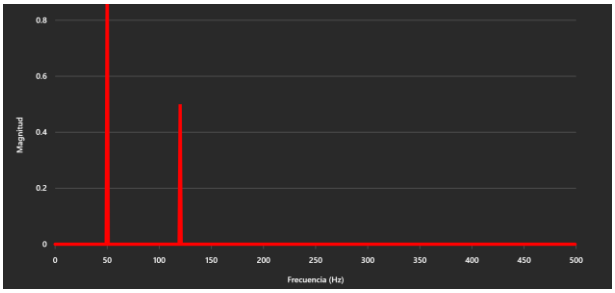

Figura 8: FFT de la señal senoidal

4 DESARROLLO

Para la metodologia de trabajo en esta 1ra Etapa, se utilizara un Notebook de Python, para probar el codigo.

Para simular todos los pasos antes mencionados, se generaran distintas señales (conocidas); de esta forma se conocera de antemano el espectro resultante y se podra contrastarlo posteriormente en la Etapa 2.

A estas señales se las muestreara, de acuerdo a los conceptos antes mencionados; para obtener la señal discreta que luego se procesara.

Esta señal se convolucionara con la respuesta al impulso de la Ventana de Hanning, para evitar discontinuidades y por lo tanto fuga espectral.

Para concluir, se le aplicara la FFT para obtener los componentes de la señal. Y con estos componentes se obtendra el valor de la THD.

Aprovechando los beneficios de Python, se graficaran todas las señales antes mencionadas para poder seguir el proceso y validar los resultados.

5 BIBLIOGRAFIA

[1] Documentación Librería Numpy [En Línea]
https://numpy.org/doc/stable/

[2] "Muestreo y ADC" – Apunte de clase para Procesamiento Embebido de Señales – Facultad Regional Avellaneda, Universidad Tecnológica Nacional

[3] "Análisis Espectral" – Apunte de clase para Procesamiento Embebido de Señales – Facultad Regional Avellaneda, Universidad Tecnológica Naciona

ANEXO

```
import tkinter as tk
from tkinter import ttk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from scipy.signal import square, sawtooth, find_peaks

# Parámetros globales

fs = 2048               # Frecuencia de muestreo (Hz)
duration = 1.0          # Duración de la señal (segundos)
t = np.linspace(0, duration, int(fs * duration), endpoint=False)  # Vector de tiempo

def generar_senal():
    # Lectura de valores desde las entradas
    amplitudes = [float(a.get() or 0) for a in amp_entries]
    frecuencias = [float(f.get() or 0) for f in freq_entries]
    tipos = [tipo.get() for tipo in type_selectors]
    duty_cycles = [duty.get() for duty in duty_sliders]

    # Inicializa señal en cero
    signal = np.zeros_like(t)

    # Construcción de la señal sumando las componentes activas
    for A, f, tipo, duty in zip(amplitudes, frecuencias, tipos, duty_cycles):
        if A > 0 and f > 0:
            if tipo == "Senoidal":
                signal += A * np.sin(2 * np.pi * f * t)
            elif tipo == "Cuadrada":
                signal += A * square(2 * np.pi * f * t, duty=duty/100)
            elif tipo == "Triangular":
                signal += A * sawtooth(2 * np.pi * f * t, width=0.5)

    # --- Análisis FFT con ventana Hanning o Blackman ---

    N = len(signal)
    window = np.hanning(N)        #Ventana Hanning
    #window = np.blackman(N)      #Ventana Blackman
    signal_windowed = signal * window
    fft_vals = np.fft.fft(signal_windowed)
    freqs = np.fft.fftfreq(N, 1/fs)
    idx = np.where(freqs >= 0)
    freqs = freqs[idx]
    amps = (2 * np.abs(fft_vals[idx])) / N
    amps = amps / np.mean(window)   # Corrección por la ventana

    # --- Detección de picos en el espectro ---

    peak_indices, _ = find_peaks(amps, height=np.max(amps)*0.05)  # picos >5% del máx
    peak_freqs = freqs[peak_indices]
    peak_amps = amps[peak_indices]

    # --- Cálculo de RMS ---

    rms_val = np.sqrt(np.mean(signal**2))
```

```
# --- Cálculo de THD (Total Harmonic Distortion) ---

    f_nonzero = [f for f in frecuencias if f > 0]
    if f_nonzero:
        f1 = min(f_nonzero)  # frecuencia fundamental
        harmonics = []
        for n in range(1, 10):
            f_target = n * f1
            idx_h = np.argmin(np.abs(freqs - f_target))
            harmonics.append(amps[idx_h])
        fundamental_amp = harmonics[0]
        other = harmonics[1:]
        thd = np.sqrt(np.sum(np.array(other)**2)) / (fundamental_amp + 1e-12)
    else:
        thd = 0.0

    # --- Limpieza de gráficos anteriores ---

    for widget in frame_signal.winfo_children(): widget.destroy()
    for widget in frame_fft.winfo_children(): widget.destroy()
    for widget in frame_info.winfo_children(): widget.destroy()

    # --- Gráfico de la señal en el tiempo ---
    fig1, ax1 = plt.subplots(figsize=(5, 2))
    ax1.plot(t, signal)
    ax1.set_title("Señal generada")
    ax1.set_xlabel("Tiempo [s]")
    ax1.set_ylabel("Amplitud [V]")
    ax1.grid(True)
    ax1.set_xlim(0, 0.1)  # se muestra solo una porción
    canvas1 = FigureCanvasTkAgg(fig1, master=frame_signal)
    canvas1.draw()
    canvas1.get_tk_widget().pack(expand=True, fill='both')

    # --- Gráfico del espectro de frecuencias ---

    fig2, ax2 = plt.subplots(figsize=(5, 2))
    ax2.plot(freqs, amps)
    ax2.set_title("Espectro")
    ax2.set_xlabel("Frecuencia [Hz]")
    ax2.set_ylabel("Amplitud [V]")
    ax2.set_xlim(0, freqs.max() * 1.05)
    ax2.grid(True)

    # --- Anotación interactiva sobre los picos ---

    annot = ax2.annotate("", xy=(0,0), xytext=(15,15), textcoords="offset points",
                         bbox=dict(boxstyle="round", fc="yellow", alpha=0.8),
                         arrowprops=dict(arrowstyle="->"))
    annot.set_visible(False)

    # Actualiza la anotación al mover el cursor

    def update_annot(event):
        if event.inaxes == ax2 and event.xdata is not None:
            distances = np.abs(peak_freqs - event.xdata)
            idx_min = np.argmin(distances)
            if distances[idx_min] < 5:  # distancia máxima para mostrar
                x, y = peak_freqs[idx_min], peak_amps[idx_min]
```

```
            # Ajuste de posición de la etiqueta según bordes
            offset_x, offset_y = 15, 15
            if x > ax2.get_xlim()[1] * 0.8: offset_x = -60
            if y > ax2.get_ylim()[1] * 0.8: offset_y = -40

            annot.xy = (x, y)
            annot.set_position((offset_x, offset_y))
            annot.set_text(f"f: {x:.2f} Hz\nA: {y:.4f} V")
            annot.set_visible(True)
            fig2.canvas.draw_idle()
        else:
            annot.set_visible(False)
            fig2.canvas.draw_idle()
    else:
        annot.set_visible(False)
        fig2.canvas.draw_idle()


# Inserta el gráfico en el frame

    canvas2 = FigureCanvasTkAgg(fig2, master=frame_fft)
    canvas2.draw()
    widget_canvas = canvas2.get_tk_widget()
    widget_canvas.pack(expand=True, fill='both')

    # Conecta el evento del mouse
    fig2.canvas.mpl_connect("motion_notify_event", update_annot)

    # --- Muestra valores numéricos ---
    tk.Label(frame_info, text=f"RMS: {rms_val:.4f} V").pack()
    tk.Label(frame_info, text=f"THD: {thd*100:.2f}%").pack()

# --- Construcción de la interfaz gráfica principal ---

root = tk.Tk()
root.title("PES - Etapa 1 - Fortunati / Martinez")

# Configuración de filas y columnas expandibles

for i in range(7):
    root.grid_columnconfigure(i, weight=1)
root.grid_rowconfigure(4, weight=1)
root.grid_rowconfigure(5, weight=1)
root.grid_rowconfigure(6, weight=1)

amp_entries = []
freq_entries = []
type_selectors = []
duty_sliders = []

# --- Encabezados ---

tk.Label(root, text="AMPLITUD").grid(row=0, column=0, padx=5, pady=5)
tk.Label(root, text="FRECUENCIA").grid(row=1, column=0, padx=5, pady=5)
tk.Label(root, text="TIPO").grid(row=2, column=0, padx=5, pady=5)
tk.Label(root, text="DUTY (%)").grid(row=3, column=0, padx=5, pady=5)
```

```
# --- Entradas de parámetros para 4 señales ---

for i in range(4):
    a_entry = tk.Entry(root, width=6)
    f_entry = tk.Entry(root, width=6)
    a_entry.grid(row=0, column=i+1, padx=5)
    f_entry.grid(row=1, column=i+1, padx=5)
    amp_entries.append(a_entry)
    freq_entries.append(f_entry)

    tipo_cb = ttk.Combobox(root, values=["Senoidal", "Cuadrada", "Triangular"], width=8)
    tipo_cb.current(0)
    tipo_cb.grid(row=2, column=i+1, padx=5)
    type_selectors.append(tipo_cb)

    duty_var = tk.IntVar(value=50)
    duty_slider = tk.Scale(root, from_=10, to=50, orient=tk.HORIZONTAL, resolution=5, variable=duty_var)
    duty_slider.grid(row=3, column=i+1, padx=5)
    duty_sliders.append(duty_var)

# --- Botón para generar la señal ---

btn = tk.Button(root, text="GENERAR", command=generar_senal)
btn.grid(row=0, column=6, rowspan=4, padx=10)

# --- Frames donde se colocan los gráficos y resultados ---
frame_signal = tk.Frame(root)
frame_signal.grid(row=4, column=0, columnspan=7, pady=10, sticky="nsew")

frame_fft = tk.Frame(root)
frame_fft.grid(row=5, column=0, columnspan=7, pady=10, sticky="nsew")

frame_info = tk.Frame(root)
frame_info.grid(row=6, column=0, columnspan=7, pady=10, sticky="nsew")

# Inicia el bucle principal de la interfaz
root.mainloop()
```