Título de la Práctica: Análisis Espectral y Ventaneo mediante Transformada Discreta

Carrera: Ingeniería en Electrónica y Telecomunicaciones
Materia: Procesamiento Digital de Señales
Profesor: Alan David Blanco Miranda **Laboratorio:** Electrónica
Fecha: 31/03/2025 **Duración:** 3 hrs.

OBJETIVO DE LA ACTIVIDAD:

- Comprender el concepto de Transformada Discreta de Fourier (TDF) y su aplicación
- Analizar el efecto del ventaneo en la resolución espectral
- Implementar diferentes ventanas de ponderación y evaluar su desempeño
- Detectar componentes frecuenciales en señales reales mediante análisis espectral

HERRAMIENTAS, MATERIAL Y/O REACTIVOS A UTILIZAR:

1. **Hardware:**
    - Generador de funciones (con capacidad hasta 5kHz)
    - Osciloscopio digital (mínimo 2 canales)
    - Arduino UNO o similar
    - Protoboard
    - Cables jumper
    - Resistencias: 2x 10kΩ, 2x 1kΩ, 2x 4.7 kΩ
    - Capacitor de 0.1µF
    - Cable USB
    - Computadora con Python y Arduino IDE instalados
2. **Software:**
    - Arduino IDE (última versión)
    - Python 3.x con las siguientes bibliotecas:
        - numpy
        - matplotlib
        - pyserial
    - Editor de código (VS Code recomendado)

PROCEDIMIENTO:

PARTE 1: SIMULACIÓN EN PYTHON

1. Realizar una simulación en Python sobre la Transformada Discreta de Fourier:
   a) Crear un archivo Python llamado tdf_simulacion.py con el siguiente código:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

# Parámetros de simulación
fs = 1000       # Frecuencia de muestreo (Hz)
T = 1           # Duración de la señal (segundos)
N = T * fs      # Número de muestras
t = np.arange(0, T, 1/fs)  # Vector de tiempo

# Crear señales de prueba
f1 = 100        # Frecuencia de la primera señal (Hz)
f2 = 110        # Frecuencia de la segunda señal (Hz)

# Caso 1: Dos frecuencias cercanas con igual amplitud
x1 = np.cos(2*np.pi*f1*t) + np.cos(2*np.pi*f2*t)

# Caso 2: Dos frecuencias cercanas con amplitudes diferentes
x2 = np.cos(2*np.pi*f1*t) + 0.1*np.cos(2*np.pi*f2*t)

# Función para calcular y mostrar el espectro
def mostrar_espectro(x, title, N_fft=None):
    if N_fft is None:
        N_fft = len(x)

    # Calcular FFT
    X = np.fft.fft(x, N_fft)
    X_mag = np.abs(X) / N_fft  # Normalizar

    # Frecuencias para el eje x
    f = np.fft.fftfreq(N_fft, 1/fs)

    # Graficar solo la mitad positiva del espectro
    plt.figure(figsize=(10, 6))
    plt.plot(f[:N_fft//2], 2*X_mag[:N_fft//2])  # Multiplicar por 2 para conservar la energía
    plt.title(title)
    plt.xlabel('Frecuencia (Hz)')
    plt.ylabel('Magnitud')
    plt.grid(True)
    plt.xlim([0, 200])  # Limitar el rango de frecuencias mostradas

# Mostrar el espectro de las señales sin ventana
mostrar_espectro(x1, 'Espectro de dos frecuencias cercanas (100 Hz y 110 Hz)')
```

```
mostrar_espectro(x2, 'Espectro de dos frecuencias cercanas con diferente amplitud')

# Definir diferentes ventanas
ventanas = {
    'Rectangular': np.ones(N),
    'Hanning': signal.windows.hann(N),
    'Hamming': signal.windows.hamming(N),
    'Blackman': signal.windows.blackman(N)
}

# Mostrar las ventanas en el dominio del tiempo
plt.figure(figsize=(10, 6))
for name, window in ventanas.items():
    plt.plot(t, window, label=name)
plt.title('Ventanas en el dominio del tiempo')
plt.xlabel('Tiempo (s)')
plt.ylabel('Amplitud')
plt.legend()
plt.grid(True)

# Aplicar las ventanas a la señal con amplitudes iguales
plt.figure(figsize=(12, 10))
for i, (name, window) in enumerate(ventanas.items(), 1):
    plt.subplot(2, 2, i)

    # Aplicar ventana
    x_windowed = x1 * window

    # Calcular FFT
    X = np.fft.fft(x_windowed)
    X_mag = np.abs(X) / N

    # Frecuencias para el eje x
    f = np.fft.fftfreq(N, 1/fs)

    # Graficar
    plt.plot(f[:N//2], 2*X_mag[:N//2])
    plt.title(f'Ventana {name}')
    plt.xlabel('Frecuencia (Hz)')
    plt.ylabel('Magnitud')
    plt.grid(True)
    plt.xlim([90, 120])  # Zoom en la región de interés

plt.tight_layout()

# Aplicar las ventanas a la señal con amplitudes diferentes
plt.figure(figsize=(12, 10))
for i, (name, window) in enumerate(ventanas.items(), 1):
    plt.subplot(2, 2, i)
```

```
    # Aplicar ventana
    x_windowed = x2 * window

    # Calcular FFT
    X = np.fft.fft(x_windowed)
    X_mag = np.abs(X) / N

    # Frecuencias para el eje x
    f = np.fft.fftfreq(N, 1/fs)

    # Graficar
    plt.plot(f[:N//2], 2*X_mag[:N//2])
    plt.title(f'Ventana {name} - Amplitudes diferentes')
    plt.xlabel('Frecuencia (Hz)')
    plt.ylabel('Magnitud')
    plt.grid(True)
    plt.xlim([90, 120])  # Zoom en la región de interés

plt.tight_layout()
plt.show()
```

2. Ejecutar el código y analizar las gráficas obtenidas para comprender:
    - La forma básica del espectro de frecuencia
    - La resolución espectral y el fenómeno de “fuga espectral” (spectral leakage)
    - La diferencia entre las diferentes ventanas

PARTE 2: DESARROLLO y MONTAJE

1. **Configuración del Generador de Funciones**
   - Encender el generador
   - Configurar:
     - Forma de onda: Senoidal
     - Frecuencia: 100 Hz
     - Amplitud: 2Vpp
     - Offset: +1V (para mantener la señal positiva)
2. **Montaje del Circuito**

```
[Generador] ---> [R1 10kΩ] ---> |punto A| ---> [R2 10kΩ] ---> GND
                  |punto A| ---> [C1 0.1µF] ---> GND
                  |punto A| ---> Arduino A0
```

3. **Verificación con Osciloscopio**
   - Conectar Canal 1 al punto A
   - Configurar:
     - Base de tiempo: 2ms/div
     - Voltaje: 500mV/div
   - Verificar que la señal:
     - Sea visible y estable
     - No exceda 5V pico a pico
     - Mantenga su forma senoidal

Parte 3: Configuración del Software

1. Código Arduino

    1. Abra Arduino IDE
    2. Copie y pegue el siguiente código:

```
const int analogPin = A0;       // Pin analógico de entrada
const unsigned long SAMPLE_PERIOD = 200; // Microsegundos entre muestras (5kHz)
const int BUFFER_SIZE = 256;    // Tamaño del buffer
int sample_buffer[BUFFER_SIZE]; // Buffer para almacenar muestras
int buffer_index = 0;           // Índice del buffer
bool buffer_full = false;       // Indicador de buffer lleno

void setup() {
  Serial.begin(115200);         // Iniciar comunicación serial
  analogReference(DEFAULT);
  delay(1000);                  // Estabilización
}

void loop() {
  static unsigned long last_sample = 0;
```

```
unsigned long current_time = micros();

// Muestrear a la frecuencia especificada
if (current_time - last_sample >= SAMPLE_PERIOD) {
  // Leer valor analógico
  int sensor_value = analogRead(analogPin);

  // Guardar en buffer
  if (buffer_index < BUFFER_SIZE) {
    sample_buffer[buffer_index] = sensor_value;
    buffer_index++;

    if (buffer_index >= BUFFER_SIZE) {
      buffer_full = true;
    }
  }

  last_sample = current_time;
 }

 // Enviar buffer completo al PC
 if (buffer_full) {
  for (int i = 0; i < BUFFER_SIZE; i++) {
    Serial.println(sample_buffer[i]);
  }

  // Reiniciar buffer
  buffer_index = 0;
  buffer_full = false;

  // Esperar antes de adquirir el siguiente conjunto de datos
  delay(500);
 }
}
```

3. Cargue el código al Arduino
4. Verifique que no hay errores

2. Código Python para Adquisición y Análisis

    1. Cree un nuevo archivo Python llamado analisis_espectral.py con el siguiente código:

```python
import serial
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from time import sleep
import os

def adquirir_datos(puerto='COM3', buffer_size=256):
    """
    Adquiere datos del Arduino
    puerto: Puerto COM del Arduino
    buffer_size: Tamaño del buffer a leer
    """
    try:
        # Configurar puerto serial
        ser = serial.Serial(puerto, 115200)
        print("Conectado al puerto:", puerto)
        sleep(2)  # Esperar inicialización

        print("Esperando datos del Arduino...")

        # Leer datos
        datos = []
        while len(datos) < buffer_size:
            if ser.in_waiting:
                try:
                    linea = ser.readline()
                    valor = int(linea.decode().strip())
                    datos.append(valor)
                except:
                    pass  # Ignorar líneas mal formadas

        ser.close()
        return np.array(datos)

    except Exception as e:
        print(f"Error al conectar con Arduino: {e}")
        return None
```

```python
def aplicar_ventana(datos, tipo_ventana='rectangular'):
    """
    Aplica una ventana específica a los datos
    """
    N = len(datos)

    if tipo_ventana.lower() == 'rectangular':
        return datos  # La ventana rectangular no modifica los datos
    elif tipo_ventana.lower() == 'hanning':
        return datos * signal.windows.hann(N)
    elif tipo_ventana.lower() == 'hamming':
        return datos * signal.windows.hamming(N)
    elif tipo_ventana.lower() == 'blackman':
        return datos * signal.windows.blackman(N)
    else:
        print("Tipo de ventana no reconocido, usando rectangular")
        return datos

def analizar_espectro(datos, fs, tipo_ventana='rectangular'):
    """
    Analiza el espectro de frecuencia de los datos
    """
    # Remover componente DC
    datos = datos - np.mean(datos)

    # Aplicar ventana
    datos_ventana = aplicar_ventana(datos, tipo_ventana)

    # Calcular FFT
    N = len(datos)
    X = np.fft.fft(datos_ventana)
    X_mag = np.abs(X) / N  # Normalizar

    # Calcular frecuencias
    f = np.fft.fftfreq(N, 1/fs)

    return f[:N//2], 2*X_mag[:N//2]  # Retornar solo la parte positiva
```

```
def guardar_resultados(datos, fs, nombre_archivo='resultados_espectrales'):
    """
    Guarda los resultados del análisis espectral en un archivo
    """
    # Crear directorio si no existe
    if not os.path.exists('resultados'):
        os.makedirs('resultados')

    # Aplicar diferentes ventanas y guardar resultados
    ventanas = ['rectangular', 'hanning', 'hamming', 'blackman']

    plt.figure(figsize=(12, 8))

    for i, ventana in enumerate(ventanas, 1):
        # Analizar espectro con la ventana actual
        f, magnitud = analizar_espectro(datos, fs, ventana)

        # Graficar
        plt.subplot(2, 2, i)
        plt.plot(f, magnitud)
        plt.title(f'Ventana {ventana.capitalize()}')
        plt.xlabel('Frecuencia (Hz)')
        plt.ylabel('Magnitud')
        plt.grid(True)

    plt.tight_layout()
    plt.savefig(f'resultados/{nombre_archivo}.png')
    plt.close()

    print(f"Resultados guardados en resultados/{nombre_archivo}.png")

def main():
    # Configuración
    PUERTO = 'COM3'  # Cambiar según su sistema
    BUFFER_SIZE = 256
    FS = 5000  # Frecuencia de muestreo (Hz) - 1/200µs = 5kHz

    # Menú de opciones
    print("=== ANÁLISIS ESPECTRAL CON TRANSFORMADA DISCRETA DE
FOURIER ===")
    print("1. Adquirir datos y analizar")
    print("2. Salir")

    opcion = input("Seleccione una opción: ")

    if opcion == '1':
        # Seleccionar puerto
        puerto = input(f"Ingrese el puerto COM (default: {PUERTO}): ") or PUERTO
```

```
        # Adquirir datos
        datos = adquirir_datos(puerto, BUFFER_SIZE)

        if datos is not None:
            print(f"Se adquirieron {len(datos)} muestras")

            # Mostrar señal en el tiempo
            plt.figure(figsize=(10, 6))
            t = np.arange(0, len(datos)/FS, 1/FS)
            plt.plot(t, datos)
            plt.title('Señal adquirida en el tiempo')
            plt.xlabel('Tiempo (s)')
            plt.ylabel('Amplitud (ADC)')
            plt.grid(True)
            plt.show()

            # Analizar y mostrar espectro con diferentes ventanas
            ventanas = ['rectangular', 'hanning', 'hamming', 'blackman']

            plt.figure(figsize=(12, 8))

            for i, ventana in enumerate(ventanas, 1):
                # Analizar espectro con la ventana actual
                f, magnitud = analizar_espectro(datos, FS, ventana)

                # Graficar
                plt.subplot(2, 2, i)
                plt.plot(f, magnitud)
                plt.title(f'Ventana {ventana.capitalize()}')
                plt.xlabel('Frecuencia (Hz)')
                plt.ylabel('Magnitud')
                plt.grid(True)

            plt.tight_layout()
            plt.show()

            # Guardar resultados
            guardar = input("¿Desea guardar los resultados? (s/n): ")
            if guardar.lower() == 's':
                nombre = input("Nombre del archivo (sin extensión): ") or
'resultados_espectrales'
                guardar_resultados(datos, FS, nombre)

        print("¡Gracias por usar el analizador espectral!")

if __name__ == "__main__":
    main()
```

3. Asegúrese de tener todas las bibliotecas necesarias instaladas:

   pip install numpy matplotlib scipy pyserial

Procedimiento Experimental

Experimento 1: Efecto de la Frecuencia de Muestreo

Experimento Base

- Configure el generador a 100 Hz
- Ejecute el programa Python y seleccione la opción para adquirir datos
- Observe y documente:
  - La forma de la señal en el tiempo
  - El espectro obtenido con cada ventana
  - La amplitud y ancho del lóbulo principal
  - La amplitud de los lóbulos secundarios

Experimento con Dos Frecuencias Cercanas

- Configure el generador para generar una señal de 100 Hz
- Adquiera y analice la señal con las distintas ventanas
- Cambie el generador a 110 Hz
- Adquiera y analice la nueva señal
- Compare visualmente los espectros obtenidos para cada frecuencia
- Calcule la resolución espectral teórica: $\Delta f = fs/N$

Experimento con Interferencia y Ruido

- Configure el generador a 100 Hz
- Añada una fuente de ruido (puede ser otro generador o simplemente acercar un teléfono móvil)
- Analice el espectro de la señal con ruido usando las diferentes ventanas
- Compare la capacidad de cada ventana para discriminar la señal del ruido

- **Análisis de Resolución Espectral**

  - Configure el generador a dos frecuencias cercanas (puede requerir un generador más avanzado)
  - Alternativamente, modifique el código Python para generar dos frecuencias mediante Arduino
  - Experimente con diferentes tamaños de buffer (N) para ver cómo afecta la resolución
  - Documente la capacidad de cada ventana para separar las frecuencias cercanas

Análisis de Resultados

Para cada experimento, analice:

1. Comparación entre ventanas:
   - ¿Qué ventana ofrece la mejor resolución espectral?
   - ¿Qué ventana tiene mejor supresión de lóbulos secundarios?
   - ¿Cuál es el compromiso entre resolución y fuga espectral?
2. Análisis del efecto del tamaño de la muestra (N):
   - ¿Cómo afecta el tamaño del buffer a la resolución espectral?
   - ¿Existe un punto óptimo entre tiempo de cómputo y calidad?
3. Evaluación de capacidad de detección:
   - ¿Qué ventana funciona mejor para detectar señales débiles en presencia de ruido?
   - ¿Qué ventana es óptima para discriminar frecuencias cercanas?
4. Aplicación del teorema de Nyquist:
   - Verifique si la frecuencia de muestreo cumple con el teorema de Nyquist
   - Relacione con el fenómeno de aliasing estudiado en prácticas previas

Preguntas de Comprensión

1. ¿Qué es la Transformada Discreta de Fourier y por qué es importante en el procesamiento digital de señales?
2. Explique el fenómeno de “fuga espectral” (spectral leakage) y cómo las ventanas de ponderación ayudan a mitigarlo.
3. Compare y contraste las ventanas Rectangular, Hanning, Hamming y Blackman en términos de:
   - Resolución espectral
   - Supresión de lóbulos secundarios
   - Aplicaciones ideales para cada una
4. ¿Cómo se relaciona la resolución en frecuencia con el número de muestras y la frecuencia de muestreo?
5. ¿Por qué el espectro de una señal discreta es periódico? Relacione esto con el teorema de muestreo.

Recomendaciones y Tips

- Verificar siempre las conexiones antes de energizar
- Tomar notas de las observaciones durante los experimentos
- Mantener el osciloscopio conectado para verificar la señal de entrada
- Ajustar la frecuencia de muestreo según las capacidades del Arduino
- Guardar todas las gráficas generadas para el reporte
- Experimentar con diferentes valores de N para ver su efecto en la resolución

Posibles Problemas y Soluciones

1. No se ve señal en el osciloscopio:
    - Verifique conexiones
    - Ajuste escala de voltaje
    - Compruebe que el generador está encendido y configurado
2. Arduino no recibe datos:
    - Verifique el divisor de voltaje
    - Asegure que el voltaje máximo en A0 no exceda 5V
    - Compruebe la conexión USB
    - Reducir la frecuencia de muestreo si hay problemas
    - Revisar el monitor serial para detectar errores
3. Las gráficas espectrales no muestran picos claros:
    - Asegurar que la señal de entrada es estable
    - Aumentar el número de muestras
    - Verificar que la relación señal/ruido es adecuada

Referencias y Recursos Adicionales

1. Documentación de Arduino:
    - Referencia AnalogRead
    - Comunicación Serial
2. Documentación de Python:
    - NumPy Documentation
    - Matplotlib Documentation
3. Teoría Transformada Discreta de Fourier:
    - https://es.wikipedia.org/wiki/Transformada_discreta_de_Fourier

- Ventanas en Procesamiento de Señales: https://en.wikipedia.org/wiki/Window_function
- Oppenheim, A. V., & Schafer, R. W. (2010). Discrete-time signal processing. Pearson.

CRITERIOS DE EVALUACIÓN

1. Implementación correcta de la adquisición de datos (20%)
2. Análisis espectral con diferentes ventanas (25%)
3. Desarrollo experimental y mediciones (25%)
4. Análisis de resultados y conclusiones (30%)

OBSERVACIONES:

- Tomar fotografías del montaje experimental
- Guardar las gráficas generadas en Python
- Documentar cualquier variación observada
- Mantener respaldo digital del código utilizado