Guía de Laboratorio 1

Fundamentos de Señales Telefónicas y Codificación de Voz en Matlab

Escuela Profesional de Ingeniería en Telecomunicaciones
Universidad Nacional de San Agustín de Arequipa

OBJETIVOS

Objetivo General: Comprender los fundamentos de las señales telefónicas, generación de tonos DTMF y técnicas de codificación de voz mediante simulaciones en Matlab.

Objetivos Específicos:

1. Generar y analizar tonos DTMF en el dominio temporal y frecuencial
2. Implementar modulación PCM con cuantización uniforme y no uniforme
3. Analizar los efectos de cuantización en la calidad de voz
4. Comparar técnicas de companding (ley A y ley u)
5. Calcular métricas de calidad de voz (SNQ, MOS)

MATERIALES

Software Requerido:

- Matlab R2018b o superior
- Toolbox de Procesamiento de Señales
- Toolbox de Comunicaciones (recomendado)

Hardware:

- Laptop o PC
- Tarjeta de sonido funcional
- Micrófono

MARCO TEÓRICO

1. Tonos DTMF (Dual-Tone Multi-Frequency)

Sistema de señalización por tonos utilizado en telefonía donde cada dígito está representado por dos tonos sinusoidales simultáneos:

$$\begin{aligned} \text{Frecuencias bajas:} &\quad 697, 770, 852, 941 \text{ Hz} \\ \text{Frecuencias altas:} &\quad 1209, 1336, 1477, 1633 \text{ Hz} \end{aligned}$$

2. Modulación PCM (Pulse Code Modulation)

Proceso de digitalización de señales analógicas:

- **Muestreo:** $f_s \geq 2f_{max}$ (Teorema de Nyquist)
- **Cuantización:** $Q = \frac{2V_{max}}{2^n - 1}$
- **Codificación:** Representación binaria

3. Companding

Técnica de compresión-expansión:

- **Ley A:** $y = \frac{A|x|}{1+\ln(A)} \quad \text{para } |x| < \frac{1}{A}$
- **Ley u:** $y = \frac{\ln(1+\mu|x|)}{\ln(1+\mu)}$

4. Métricas de Calidad

- **SNQ:** $\text{SNQ} = 10 \log_{10}\left(\frac{P_{senal}}{P_{error}}\right)$
- **MOS:** Escala subjetiva 1-5

DESARROLLO DE LABORATORIO

PARTE 1: GENERACIÓN Y ANÁLISIS DE TONOS DTMF

```
%% LABORATORIO 1-2: SISTEMAS DE TELEFONÍA
clear all; close all; clc;

%% 1.1. PARÁMETROS GENERALES
fs = 8000;              % Frecuencia de muestreo (Hz)
duracion = 0.5;         % Duración de cada tono (segundos)
t = 0:1/fs:duracion;    % Vector de tiempo

% Frecuencias DTMF (Hz)
frecuencias_bajas = [697, 770, 852, 941];
frecuencias_altas = [1209, 1336, 1477, 1633];
```

```
%% 1.2. GENERACIÓN DE TONOS DTMF PARA DÍGITO '5'
f_baja = frecuencias_bajas(2);  % 770 Hz
f_alta = frecuencias_altas(2);  % 1336 Hz

% Generación de tonos
tono_bajo = sin(2*pi*f_baja*t);
tono_alto = sin(2*pi*f_alta*t);
senal_dtmf = tono_bajo + tono_alto;

% Normalización
senal_dtmf = senal_dtmf / max(abs(senal_dtmf));

%% 1.3. VISUALIZACIÓN EN DOMINIO TEMPORAL
figure('Position', [100, 100, 1200, 800]);
subplot(3,2,1);
plot(t(1:500), senal_dtmf(1:500), 'b', 'LineWidth', 1.5);
title(['Señal DTMF para dígito 5 (', num2str(f_baja), ' Hz + ', ...
       num2str(f_alta), ' Hz)']);
xlabel('Tiempo (s)'); ylabel('Amplitud'); grid on; xlim([0, 0.05]);

%% 1.4. ANÁLISIS EN DOMINIO FRECUENCIAL
N = length(senal_dtmf);
f = (-N/2:N/2-1)*(fs/N);  % Vector de frecuencias

% Transformada de Fourier
espectro = fftshift(fft(senal_dtmf));
espectro_abs = abs(espectro)/max(abs(espectro));

subplot(3,2,2);
plot(f, espectro_abs, 'r', 'LineWidth', 1.5);
title('Espectro de frecuencia de la señal DTMF');
xlabel('Frecuencia (Hz)'); ylabel('Amplitud Normalizada');
grid on; xlim([0, 2000]);

%% 1.5. GENERACIÓN DE TONOS PARA TODOS LOS DÍGITOS
tonos_dtmf = cell(4,4);
for i = 1:4
    for j = 1:4
        f_baja = frecuencias_bajas(i);
        f_alta = frecuencias_altas(j);

        tono_bajo = sin(2*pi*f_baja*t);
        tono_alto = sin(2*pi*f_alta*t);
        senal = (tono_bajo + tono_alto) / 2;

        tonos_dtmf{i,j} = senal;
    end
```

```
end

% Reproducir tono del dígito 5
sound(senal_dtmf, fs);
pause(duracion + 0.5);
```

PARTE 2: MODULACIÓN PCM Y CODIFICACIÓN DE VOZ

```
%% 2.1. CARGA DE SEÑAL DE VOZ DE PRUEBA
try
    [voz_original, fs_voz] = audioread('voz_prueba.wav');
catch
    % Generar señal de voz sintética si no hay archivo
    fs_voz = 8000;
    t_voz = 0:1/fs_voz:2;
    voz_original = 0.5*sin(2*pi*500*t_voz) + 0.3*sin(2*pi*1200*t_voz);
    voz_original = voz_original(:);
end

% Normalizar y recortar
voz_original = voz_original(1:min(16000, length(voz_original)));
voz_original = voz_original / max(abs(voz_original));

%% 2.2. MODULACIÓN PCM CON CUANTIZACIÓN UNIFORME
function [voz_cuantizada, niveles] = pcm_uniforme(senal, bits)
    niveles = 2^bits;
    paso = 2 / (niveles - 1);
    voz_cuantizada = round(senal / paso) * paso;
end

% Aplicar PCM uniforme con diferentes resoluciones
bits = [4, 8, 12];
voz_pcm = cell(length(bits), 1);

for i = 1:length(bits)
    [voz_pcm{i}, niveles] = pcm_uniforme(voz_original, bits(i));

    subplot(3,2,2+i);
    plot((1:500)/fs_voz, voz_original(1:500), 'b', 'LineWidth', 1);
    hold on;
    plot((1:500)/fs_voz, voz_pcm{i}(1:500), 'r', 'LineWidth', 1.5);
    title(['PCM Uniforme - ', num2str(bits(i)), ' bits']);
    xlabel('Tiempo (s)'); ylabel('Amplitud');
    legend('Original', 'Cuantizada'); grid on;
end

%% 2.3. COMPANDING - LEY A
function senal_compandida = companding_leyA(senal, A)
```

```
    senal_compandida = zeros(size(senal));
    for n = 1:length(senal)
        if abs(senal(n)) < 1/A
            senal_compandida(n) = A * abs(senal(n)) / (1 + log(A));
        else
            senal_compandida(n) = (1 + log(A * abs(senal(n)))) / (1 + log(A));
        end
        senal_compandida(n) = sign(senal(n)) * senal_compandida(n);
    end
end

A = 87.6;
voz_compandida = companding_leyA(voz_original, A);

% Aplicar PCM después de companding
[voz_pcm_compandida, ~] = pcm_uniforme(voz_compandida, 8);

%% 2.4. CÁLCULO DE SNQ (Señal a Ruido de Cuantización)
function snq = calcular_snq(original, cuantizada)
    error = original - cuantizada;
    potencia_senal = mean(original.^2);
    potencia_error = mean(error.^2);
    snq = 10 * log10(potencia_senal / potencia_error);
end

% Comparar SNQ para diferentes configuraciones
snq_uniforme_8bit = calcular_snq(voz_original, voz_pcm{2});
snq_compand_8bit = calcular_snq(voz_original, voz_pcm_compandida);

fprintf('SNQ PCM Uniforme 8-bit: %.2f dB\n', snq_uniforme_8bit);
fprintf('SNQ PCM + Companding A-law 8-bit: %.2f dB\n', snq_compand_8bit);

%% 2.5. REPRODUCCIÓN COMPARATIVA
disp('Reproduciendo voz original...');
sound(voz_original, fs_voz);
pause(3);

disp('Reproduciendo voz con PCM uniforme 8-bit...');
sound(voz_pcm{2}, fs_voz);
pause(3);

disp('Reproduciendo voz con companding A-law...');
sound(voz_pcm_compandida, fs_voz);

%% 2.6. ANÁLISIS ESPECTROCOMPARATIVO
figure('Position', [100, 100, 1200, 600]);

% Espectro original
```

```
subplot(2,2,1);
N = length(voz_original);
f = (0:N-1)*(fs_voz/N);
espectro_orig = 20*log10(abs(fft(voz_original)));
plot(f(1:N/2), espectro_orig(1:N/2));
title('Espectro - Voz Original');
xlabel('Frecuencia (Hz)'); ylabel('Amplitud (dB)'); grid on;

% Espectro PCM uniforme
subplot(2,2,2);
espectro_pcm = 20*log10(abs(fft(voz_pcm{2})));
plot(f(1:N/2), espectro_pcm(1:N/2));
title('Espectro - PCM Uniforme 8-bit');
xlabel('Frecuencia (Hz)'); ylabel('Amplitud (dB)'); grid on;

% Espectro error de cuantización
subplot(2,2,3);
error = voz_original - voz_pcm{2};
espectro_error = 20*log10(abs(fft(error)));
plot(f(1:N/2), espectro_error(1:N/2));
title('Espectro - Error de Cuantización');
xlabel('Frecuencia (Hz)'); ylabel('Amplitud (dB)'); grid on;

% Espectro con companding
subplot(2,2,4);
espectro_comp = 20*log10(abs(fft(voz_pcm_compandida)));
plot(f(1:N/2), espectro_comp(1:N/2));
title('Espectro - PCM + Companding A-law');
xlabel('Frecuencia (Hz)'); ylabel('Amplitud (dB)'); grid on;

%% 2.7. ESTIMACIÓN DE MOS (MODELO SIMPLIFICADO)
function mos = estimar_mos(snq)
    % Modelo E-model simplificado para estimar MOS
    if snq > 35
        mos = 4.5;
    elseif snq > 30
        mos = 4.0;
    elseif snq > 25
        mos = 3.5;
    elseif snq > 20
        mos = 3.0;
    else
        mos = 2.5;
    end
end

mos_uniforme = estimar_mos(snq_uniforme_8bit);
mos_compand = estimar_mos(snq_compand_8bit);
```

```
fprintf('\n--- CALIDAD DE VOZ ESTIMADA ---\n');
fprintf('PCM Uniforme 8-bit: MOS = %.1f\n', mos_uniforme);
fprintf('PCM + Companding: MOS = %.1f\n', mos_compand);
```

PROGRAMA PARA CREAR ARCHIVO voz_prueba.wav

```
%% GRABADORA SIMPLE DE VOZ
% Crea el archivo voz_prueba.wav desde el micrófono

clear; close all; clc;

%% Configuración
fs = 8000;          % Frecuencia de muestreo (8 kHz)
duracion = 5;       % Duración de la grabación en segundos
nombre_archivo = 'voz_prueba.wav';

fprintf('=== GRABADORA DE VOZ ===\n');
fprintf('Frecuencia: %d Hz\n', fs);
fprintf('Duración: %d segundos\n', duracion);
fprintf('Archivo: %s\n\n', nombre_archivo);

%% Grabación desde micrófono
fprintf('Preparando grabación...\n');

try
    % Crear objeto de grabación
    grabadora = audiorecorder(fs, 16, 1); % 16 bits, mono

    fprintf('Grabando durante %d segundos...\n', duracion);
    fprintf('HABLE AHORA...\n');

    % Iniciar grabación
    recordblocking(grabadora, duracion);

    % Obtener los datos de audio
    audio_data = getaudiodata(grabadora);

    fprintf('Grabación completada.\n');

catch error
    fprintf('Error: No se pudo acceder al micrófono.\n');
    fprintf('Mensaje: %s\n', error.message);
    return;
end

%% Normalizar audio
audio_data = audio_data / max(abs(audio_data)) * 0.99;
```

```
%% Guardar archivo WAV
try
    audiowrite(nombre_archivo, audio_data, fs);
    fprintf('Archivo guardado: %s\n', nombre_archivo);

    % Mostrar información del archivo
    info = audioinfo(nombre_archivo);
    fprintf('Duración: %.2f segundos\n', info.Duration);
    fprintf('Muestras: %d\n', info.TotalSamples);
    fprintf('Tamaño: %.2f KB\n', info.FileSize/1024);

catch error
    fprintf('Error al guardar el archivo: %s\n', error.message);
    return;
end

%% Reproducir grabación
fprintf('\n¿Reproducir grabación? (s/n): ');
respuesta = input('', 's');

if lower(respuesta) == 's'
    fprintf('Reproduciendo...\n');
    sound(audio_data, fs);
    pause(duracion + 1);
end

fprintf('\n=== PROCESO COMPLETADO ===\n');
fprintf('El archivo %s está listo para usar.\n', nombre_archivo);
```

INSTRUCCIONES FINALES

1. **Ejecutar** cada sección del código paso a paso
2. **Documentar** los resultados obtenidos en cada etapa
3. **Experimentar** modificando parámetros (fs, bits, A-law)
4. **Elaborar** un informe en formato IEEE (doble columna) con los siguientes apartados:
    - Introducción
    - Marco teórico
    - Desarrollo
    - Resultados
    - Conclusiones