UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA, SEDE CENTRAL

INGENIERIA EN SISTEMAS

BRYAN BARRIOS

MÉTODOS NUMÉRICOS

MODIFICADOR DE VOZ CON MATLAB

INTEGRANTES:

Mario Alejandro Corzo Peralta – 0901-23-11238

Nills Berducido Gómez – 0901-23-14275

Ronald Alexis Morales Varela – 0901-23-6114

Guatemala 17 de mayo de 2025

INTRODUCCIÓN

El procesamiento de señales de audio es un campo clave en la ingeniería electrónica, informática y musical. En este proyecto se desarrolla la aplicación en MATLAB mediante App Designer para modificar una señal de voz o sonido. Esta app permite aplicar filtros digitales, eco y cambio de tono, conocido como pitch, además de visualizar los resultados. El sistema se basa en métodos numéricos como interpolación lineal, algoritmos de filtrado digital (Butterworth), y manipulación temporal mediante desplazamiento de muestras. Además de su relevancia en áreas técnicas, también es fundamental en la industria del entrenamiento, la comunicación digital y la inteligencia artificial. El análisis y la manipulación de audio han cobrado un papel importante en el desarrollo de sistemas inteligentes, tales como asistentes de voz, sistemas de reconocimiento de habla y generación automática de audio.

MATLAB se posiciona como una herramienta poderosa para la implementación de algoritmos de procesamiento de señales debido a su extensa biblioteca de funciones, su capacidad de visualización gráfica y su entorno de desarrollo altamente integrado. Utilizando App Designer, es posible desarrollar interfaces interactivas que mejoran la experiencia del usuario y permiten una aplicación práctica mediante de conceptos teóricos.

El presente proyecto busca no solo demostrar la implementación técnica de modificador de voz, sino también resaltar su aplicabilidad real en productos comerciales y educativos, integrando conocimientos de programación, análisis de señales, diseño de interfaz gráfica y algoritmos numéricos. Este enfoque integral permite al estudiante o profesional abordar problemas reales mediante soluciones computacionales eficientes y versátiles.

OBJETIVOS

General:

1. Desarrollar una aplicación interactiva en MATLAB que modifique señales de audio aplicando diversos efectos sonoros como filtro, eco, pitch shifting y permitan visualizar los resultados.
2. Aplicar métodos numéricos y algoritmos de señal para manipular características del audio como frecuencia, eco y tono.
3. Proporcionar una herramienta educativa que permita visualizar los efectos de procesamiento digital en el dominio del tiempo y la frecuencia.

Específicos:

1. Aplicar un filtro digital pasabajos utilizando el algoritmo de Butterworth.
2. Simular eco mediante desplazamiento de muestras y mezcla con la señal original.
3. Realizar cambio de tono mediante interpolación lineal.
4. Visualizar la forma de onda y el espectrograma del audio procesado.

JUSTIFICACIÓN

En el campo del procesamiento de señales, este tipo de herramientas es útil tanto para la educación como para el desarrollo de soluciones prácticas. El procesamiento en tiempo casi real de señales de voz tiene aplicaciones en:

- Mejora de calidad de audio en telecomunicaciones.
- Desarrollo de asistentes de voz, interfaces auditivas y efectos musicales.
- Seguridad: Enmascaramiento o distorsión de voz para anonimato.

Además, el uso de MATLAB facilita el prototipado rápido de estas aplicaciones con interfaz gráfica intuitiva, permitiendo su comprensión por estudiantes y profesionales. Esta problemática se vuelve aun mas relevante con el crecimiento de tecnologías portátiles y móviles, donde el procesamiento de voz en tiempo real es esencial para asistentes personales, traducción automática y sistemas de comunicación manos libres.

El desarrollo de este proyecto no solo fortalece habilidades técnicas como el análisis de señales y la programación en MATLAB, sino también capacidades de diseño de interfaces gráficas y resolución de problemas. Al integrar herramientas visuales con algoritmos numéricos, se promueve la comprensión mas profunda del comportamiento de los sistemas de procesamiento digital.

Además, permite el desarrollo de prototipos funcionales que podrían escalarse para soluciones comerciales o de investigación. Por lo tanto, su estudio es fundamental para ingenieros en electrónica, telecomunicaciones, informática y ramas a fines que buscan proponer innovaciones prácticas en productos tecnológicos centrados en la interacción por voz.

MARCO TEORICO:

El procesamiento digital de señales consiste en la manipulación matemática de señales en forma digital. En este proyecto, se procesan señales de audio mediante técnicas de filtrado, adición de eco y cambio de pitch.

Filtro Digital Pasa Bajos: Los filtros pasa bajos eliminan frecuencias altas no deseadas, conservando el contenido frecuencial más bajo. Se utiliza un filtro Butterworth por su respuesta suave y sin ondulaciones en la banda pasante.

Eco Digital: El eco se genera al sumar a la señal original una copia de sí misma con cierto retardo y atenuación. Este método es común en simulaciones acústicas.

Pitch Shifting: Cambiar el tono implica comprimir o expandir la duración temporal de la señal, lo cual se logra mediante interpolación de las muestras.

Ecuaciones y Modelos Matemáticos

1. Filtro Butterworth Pasa Bajos:

- Diseño del filtro:

$$[b, a] = butter(n, \frac{fc}{\frac{fs}{2}})$$

- Aplicación del filtro:

$$y[n] = \Sigma\,(b_k * x[n-k]) - \Sigma\,(a_k * y[n-k])$$

2. Eco (Delay Line):

$$y[n] = x[n] + \alpha * x[n - D]$$

1. **Interpolación Lineal (Pitch Shifting):**

$$y[k] = x[i] + (x[i+1] - x[i]) * (k - i)$$

donde i = floor(k / r)

<u>Métodos Numéricos Aplicados</u>

Interpolación Lineal:

Se define un nuevo vector de tiempo con una densidad de muestras ajustada al factor de pitch. Se interpola entre cada par de muestras originales para obtener las nuevas muestras.

Fórmula clave:

$$y(t) = x(t_0) + \left(\frac{(x(t_1) - x(t_0))}{(t_1 - t_0)}\right) * (t - t_0)$$

Filtro Digital (Butterworth):

1. Se definen coeficientes con butter(n, fc).
2. Se aplica el filtrado con filter(b, a, x).

Desplazamiento de Muestras (Delay Line):

1. Se crea una copia de la señal original desplazada por D muestras.
2. Se suma con la señal original multiplicada por un coeficiente de mezcla.

Desarrollo de la Interfaz en App Designer

La aplicación fue desarrollada utilizando App Designer. Esta interfaz permite una interacción directa con el procesamiento del audio sin necesidad de modificar el código fuente. Componentes utilizados:

- Botones para cargar, aplicar efectos, reproducir y guardar.
- Sliders para frecuencia de corte, eco y pitch.

- UIAxes para graficar la forma de onda y espectrograma.

Esta interfaz facilita el flujo de trabajo y mejora la experiencia del usuario final.

Análisis de Resultados

El filtro pasa bajo elimina las frecuencias altas, suavizando el sonido. El eco genera una reverberación que añade profundidad. El cambio de pitch modifica la percepción tonal de la voz.

Visualmente:

- La forma de onda permite observar los cambios temporales, como repeticiones por el eco.

- El espectrograma muestra la reducción de frecuencias altas y los desplazamientos espectrales causados por el cambio de pitch.

Funciones Implementadas

Para modularizar el código, se crearon funciones auxiliares:

- agregar_eco(): aplica retardo y mezcla con coeficiente.

- filtro_pasa_bajos(): diseña y aplica un filtro Butterworth.

- pitch_shift(): interpola la señal para cambiar el tono.

Estas funciones permiten una implementación más clara, mantenible y escalable del sistema.

Modificador de Voz con Matlab App Designer

Este proyecto consiste en el desarrollo de una aplicación interactiva que permite modificar señales de audio mediante la aplicación de efectos como filtro pasa bajos, eco e interpolación lineal para cambio de pitch (tono). Adicionalmente, permite visualizar la señal modificada en el dominio del tiempo y en el dominio de la frecuencia mediante un espectrograma.

Todo el procesamiento se realiza en MATLAB, aprovechando el entorno gráfico de App Designer, lo que permite una experiencia intuitiva para el usuario final.

1. **Entorno:**

• Lenguaje: MATLAB

• Interfaz: App Designer (.mlapp)

• Formato: aplicación gráfica con botones, sliders y gráficos.

2. **Estructura de la app:**

• Se usaron sliders para controlar los parámetros de los efectos (frecuencia de corte, eco y pitch).

• Se agregaron botones para cargar el audio, aplicar los efectos, reproducirlo y guardarlo.

• Se utilizaron dos ejes (UIAxes) para graficar la forma de onda y el espectrograma.

3. **Procesamiento de audio:**

• Lectura del archivo .wav con audioread.

• Aplicación de un filtro digital pasa bajos con butter y filter.

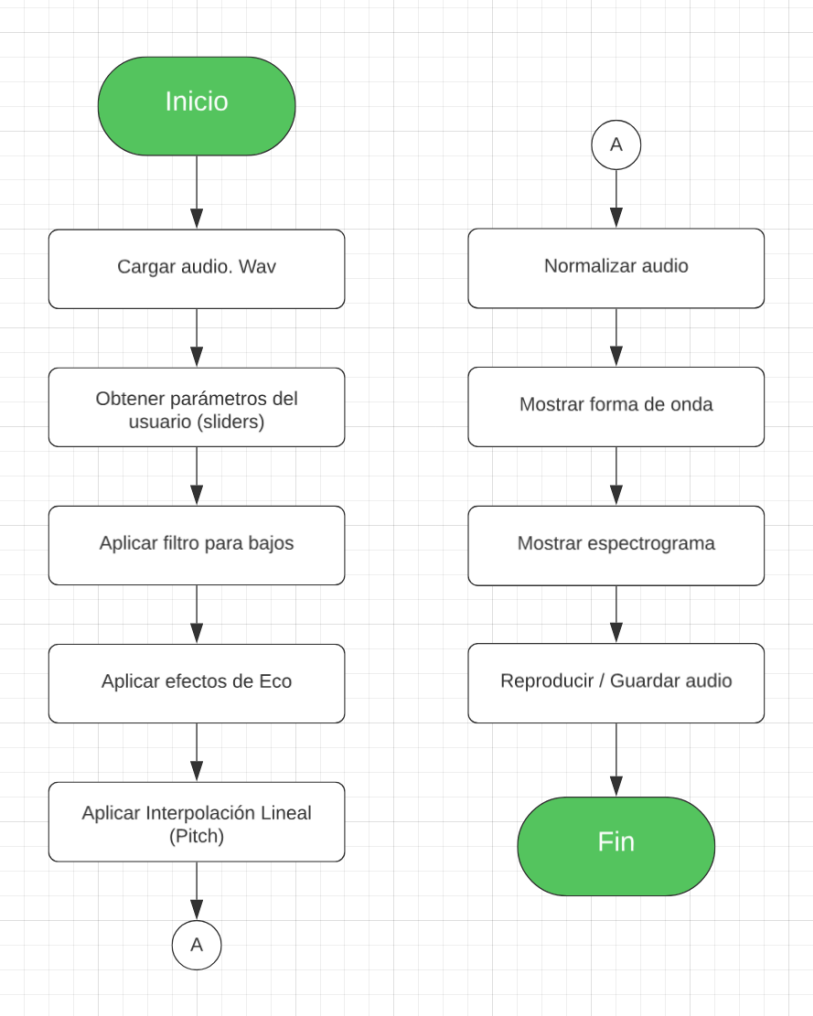
• Generación de eco mediante desplazamiento de muestras.

• Aplicación de interpolación lineal para cambio de duración o tono.

• Visualización de espectro mediante spectrogram.

IMLEMENTACIÓN EN MATLAB:

Diagrama de Flujo



Datos de Entrada

1 Un archivo de audio en formato .wav (mono o estéreo).

2 Parámetros del usuario mediante sliders:

• Frecuencia de corte del filtro pasa bajos (500 Hz a 5000 Hz).

• Intensidad del eco (0 a 1).

• Factor de pitch (0.5 para grave, 2.0 para agudo).

Funcionalidad

La aplicación permite:

1. Cargar una señal de audio desde el disco.
2. Visualizar la forma de onda original.
3. Aplicar procesamiento digital:

   •Filtro Butterworth pasa bajos.

   •Eco con retardo e intensidad ajustable.

   •Pitch shifting usando interpolación lineal.

4. Mostrar resultados gráficamente.

5. Reproducir el audio modificado directamente desde la app.

6. Guardar el archivo de audio final.

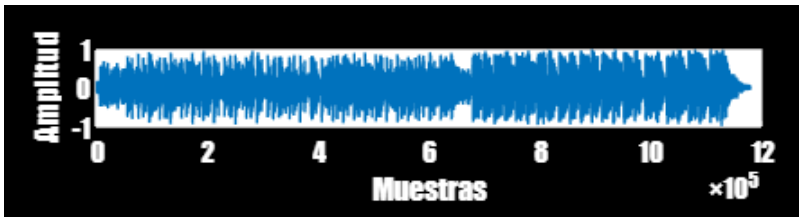
Todo el procesamiento se hace en tiempo casi real, permitiendo ajustes inmediatos por el usuario.

Gráficos

Forma de onda del audio modificado.

• Muestra la amplitud en función de las muestras (dominio del tiempo).

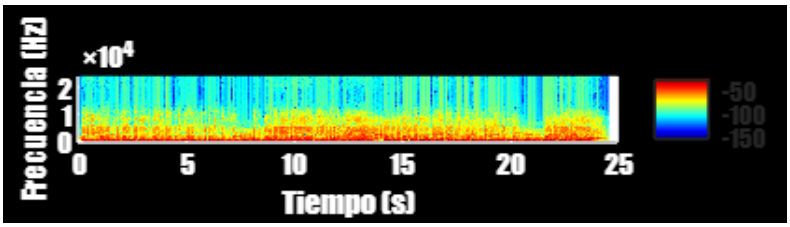
• Permite observar cambios causados por los efectos.



Espectrograma.

• Muestra cómo varían las frecuencias en el tiempo.

• Permite ver la eliminación de frecuencias (por el filtro) o la modificación del tono (por el pitch).



Uso de Funciones

La app puede estructurarse con funciones auxiliares para modularizar el código:

Función agregar_eco

```
% --- Eco ---
delay_segundos = 0.3;
delay_muestras = round(delay_segundos * Fs);
if delay_muestras > length(audio)
    delay_muestras = length(audio) - 1;
end
if delay_muestras > 0 && abs(alpha) > 0
    audio_eco = audio;
    audio_eco(delay_muestras+1:end) = ...
        audio_eco(delay_muestras+1:end) + alpha * audio(1:end - delay_muestras);
    audio = audio_eco;
end
```

Función filtro_pasa_bajos

```
% --- Filtro pasa bajos con protección ---
if Fc >= Fs / 2
    Fc = (Fs / 2) - 1;
elseif Fc < 1
    Fc = 1;
end

try
    [b, a] = butter(6, Fc / (Fs / 2));
    audio = filter(b, a, audio);
catch
    warning('Error en filtro pasa bajos. Se omite el filtro.');
end
```

Función pitch_shift

```
% --- Interpolación lineal adicional (antes de pitch shift) ---
factor_interp = 1.25;  % Puedes modificar este factor
x_original = 1:length(audio);
x_nuevo = linspace(1, length(audio), round(length(audio) * factor_interp));
audio_interp = interp1(x_original, audio, x_nuevo, 'linear');
audio_interp(isnan(audio_interp)) = 0;

% --- Pitch Shift con interpolación ---
if pitch_factor < 0.01
    pitch_factor = 0.01;
elseif pitch_factor > 10
    pitch_factor = 10;
end

t_original = linspace(0, length(audio_interp)/Fs, length(audio_interp));
t_nuevo = linspace(0, t_original(end), round(length(audio_interp)/pitch_factor));
audio_modificado = interp1(t_original, audio_interp, t_nuevo, 'linear');
audio_modificado(isnan(audio_modificado)) = 0;

% --- Normalizar salida ---
audio_modificado = audio_modificado / max(abs(audio_modificado) + 1e-6);

app.audio_modificado = audio_modificado;
```

Fotos de la interfaz

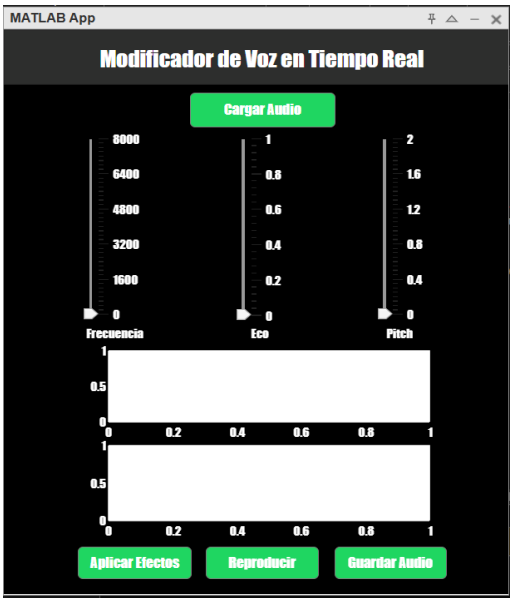


Imagen del programa ejecutado

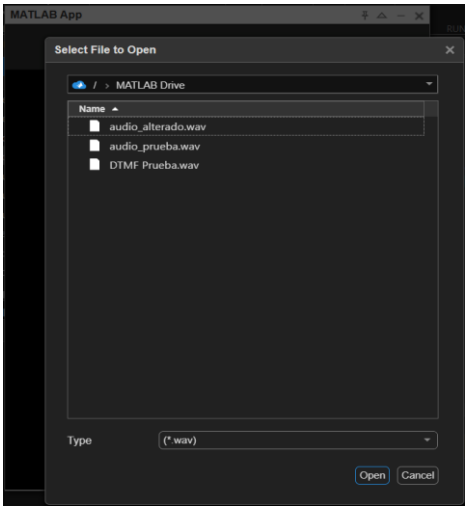


Imagen del programa al presionar el botón cargar audio.

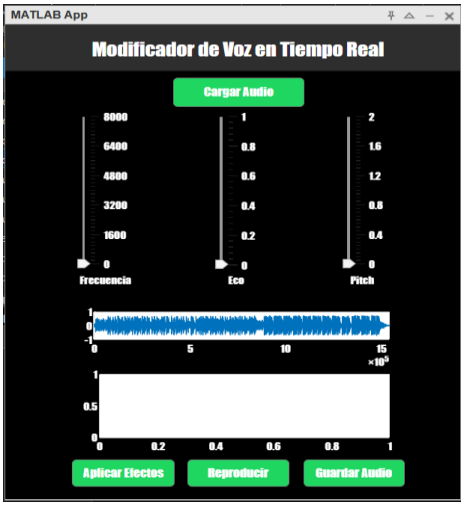
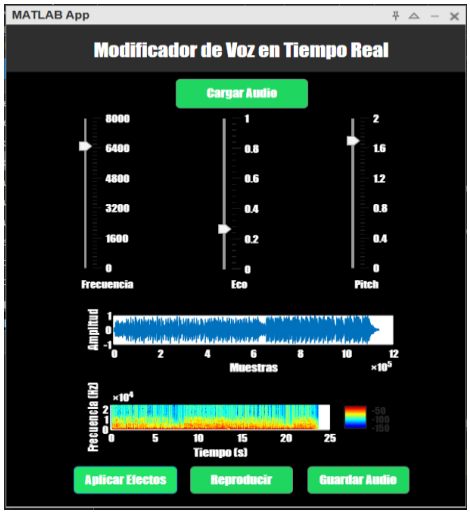


Imagen del programa con el audio ya cargado



Comparativa con MATLAB (como prototipo)

Aspecto	MATLAB Prototipo	Adobe Audition
Requiere Programación	Si	No
Tiempo Real	Limitado	En tiempo real
Interfaz de usuario	Personalizada App Designer	Profesional e intuitiva
Exportación de audio	Básica (.wav)	Multiples formatos y calidad
Integración profesional	No nativa	Integrada (Adobe CC)

Expansión futura

Integración de Inteligencia Artificial:

- Aplicar modelos de IA para la detección automática de ruido, selección del tipo de efecto más apropiado según el perfil de voz, y ajuste inteligente de parámetros.
- Incorporar servicios de IA como Adobe Sensei o modelos de voz generativos (por ejemplo, clonación de voz y mejora de habla automática).

Plataforma web o aplicación móvil:

- Ofrecer una plataforma en la nube donde los usuarios puedan subir archivos, elegir efectos y recibir la edición lista en minutos.
- Desarrollar una app móvil sencilla para grabar voz, aplicar efectos y exportar directamente a redes sociales o para doblaje rápido.

API y automatización por lotes:

- Desarrollar una API REST para que empresas o freelancers integren el procesamiento de voz en sus flujos de trabajo (por ejemplo, subir 100 audios y obtenerlos editados automáticamente).
- Automatización con Adobe Bridge, Watch Folders y acciones preconfiguradas que permitan la edición masiva sin intervención manual.

CONCLUSIONES

Este proyecto demuestra cómo MATLAB y App Designer permiten desarrollar herramientas interactivas efectivas para el procesamiento digital de señales. Se aplicaron métodos numéricos relevantes para lograr resultados auditivos y visuales significativos.

La interfaz gráfica y el código modular permiten extender esta aplicación fácilmente para nuevas funcionalidades. Es una base excelente para aplicaciones educativas, académicas o de desarrollo profesional.

ANEXOS

Fotos del código

```
        function CargarAudioButtonPushed(app, event)
            [file, path] = uigetfile('*.wav');
    if isequal(file,0)
        return;
    end
    [audio, Fs] = audioread(fullfile(path, file));
    app.audio_original = mean(audio, 2); % Convertir a mono
    app.Fs = Fs;
    plot(app.UIAxes, app.audio_original);
    title(app.UIAxes, 'Forma de Onda');
        end
```

Imagen del código del botón cargar audio

```
        function AplicarEfectosButtonPushed(app, event)
    if isempty(app.audio_original)
        uialert(app.UIFigure, 'Carga un audio primero.', 'Error');
        return;
    end

    audio = app.audio_original;
    Fs = app.Fs;

    % Obtener parámetros de sliders
    Fc = abs(app.SliderFrecuenciaCorte.Value); % Frecuencia de corte
    alpha = app.SliderIntensidadEco.Value;     % Intensidad de eco
    pitch_factor = abs(app.SliderPitchShift.Value); % Factor de pitch shift

    % --- Filtro pasa bajos con protección ---
    if Fc >= Fs / 2
        Fc = (Fs / 2) - 1;
    elseif Fc < 1
        Fc = 1;
    end

    try
        [b, a] = butter(6, Fc / (Fs / 2));
        audio = filter(b, a, audio);
    catch
        warning('Error en filtro pasa bajos. Se omite el filtro.');
    end

    % --- Eco ---
    delay_segundos = 0.3;
    delay_muestras = round(delay_segundos * Fs);
    if delay_muestras > length(audio)
        delay_muestras = length(audio) - 1;
    end
    if delay_muestras > 0 && abs(alpha) > 0
        audio_eco = audio;
        audio_eco(delay_muestras+1:end) = ...
            audio_eco(delay_muestras+1:end) + alpha * audio(1:end - delay_muestras);
```

```
        audio = audio_eco;
    end

    % --- Interpolación lineal adicional (antes de pitch shift) ---
    factor_interp = 1.25;  % Puedes modificar este factor
    x_original = 1:length(audio);
    x_nuevo = linspace(1, length(audio), round(length(audio) * factor_interp));
    audio_interp = interp1(x_original, audio, x_nuevo, 'linear');
    audio_interp(isnan(audio_interp)) = 0;

    % --- Pitch Shift con interpolación ---
    if pitch_factor < 0.01
        pitch_factor = 0.01;
    elseif pitch_factor > 10
        pitch_factor = 10;
    end

    t_original = linspace(0, length(audio_interp)/Fs, length(audio_interp));
    t_nuevo = linspace(0, t_original(end), round(length(audio_interp)/pitch_factor));
    audio_modificado = interp1(t_original, audio_interp, t_nuevo, 'linear');
    audio_modificado(isnan(audio_modificado)) = 0;

    % --- Normalizar salida ---
    audio_modificado = audio_modificado / max(abs(audio_modificado) + 1e-6);

    app.audio_modificado = audio_modificado;

    % --- Mostrar waveform ---
    cla(app.UIAxes);
    plot(app.UIAxes, audio_modificado);
    title(app.UIAxes, 'Audio Modificado');
    xlabel(app.UIAxes, 'Muestras');
    ylabel(app.UIAxes, 'Amplitud');
```

```
% --- Mostrar espectrograma ---
cla(app.UIAxes2);
[~, F, T, P] = spectrogram(audio_modificado, 256, [], [], Fs, 'yaxis');
P_dB = 10 * log10(P + eps);
imagesc(app.UIAxes2, T, F, P_dB);
axis(app.UIAxes2, 'xy');
xlabel(app.UIAxes2, 'Tiempo (s)');
ylabel(app.UIAxes2, 'Frecuencia (Hz)');
title(app.UIAxes2, 'Espectrograma');
colormap(app.UIAxes2, jet);
colorbar(app.UIAxes2);
    end
```

Imagen del código del botón aplicar efectos

```
function ReproducirButtonPushed(app, event)
if isempty(app.audio_modificado)
uialert(app.UIFigure, 'Aplica los efectos primero.', 'Error');
return;
end

% Reproducir el audio modificado
sound(app.audio_modificado, app.Fs); % Asegúrate de que `Fs` sea el de la grabación original
end
```

Imagen del código del botón reproducir

```
function GuardarAudioButtonPushed(app, event)
     if isempty(app.audio_modificado)
uialert(app.UIFigure, 'Aplica los efectos primero.', 'Error');
return;
     end
[file, path] = uiputfile('*.wav');
if isequal(file,0)
return;
end
audiowrite(fullfile(path, file), app.audio_modificado, app.Fs);
uialert(app.UIFigure, 'Audio guardado exitosamente.', 'Éxito');
end
```

Imagen del código del botón guardar audio