

ANEXO VI. Código fuente del sintetizador de timple mediante síntesis digital.

Este anexo recoge los códigos de Matlab implementados para la creación del sintetizador de timple mediante síntesis digital.

Código fuente 1:

Código sintetizador de una sola nota a partir de grabación de audio.

```
%% Listar dispositivos MIDI disponibles

midiDevices = mididevinfo;

if isempty(midiDevices.input)

    error('No se encontraron dispositivos MIDI de entrada.');
```

```

end

%% Crear objeto para la reproducción del sonido

fs = 44100; % Frecuencia de muestreo

audioOut = audioDeviceWriter('SampleRate', fs);

%% Inicializar variables

isNoteOn = false;

note = 0;

frequency = 0;

amplitude = 1;

duration = 1; % Duración de la nota en segundos

%% LECTURA MIDI

fprintf('Presiona una tecla en tu controlador MIDI.\n');

% Función para convertir número de nota MIDI a frecuencia

midiToFreq = @(note) 440 * 2.^((note - 69) / 12);

%bucle infinito esperando notas

while true

    % Leer mensajes MIDI

    midiMsg = midireceive(midiIn);

    % Procesar mensajes MIDI si hay alguno

    if ~isempty(midiMsg)

        for i = 1:length(midiMsg)

            msg = midiMsg(i);

            % Comprobar si el mensaje es Note On

            if strcmp(msg.Type, 'NoteOn') && msg.Velocity > 0

                isNoteOn = true;

                note = msg.Note;

                amplitude = msg.Velocity / 127; % Normalizar la
                velocidad a [0, 1]

                fprintf('Note On: %d, Frequency: %.2f Hz, Velocity:
                %d\n', note, frequency, msg.Velocity);

                % Generar y reproducir sonido sintetizado

                [synthesizedSound] =

```

```

synthesizeSoundFromRecording("1ABH_forte_Estéreo#E3.aif",frequency,amplitude
,duration,fs);

Velocity 0

% Enviar el sonido al dispositivo de audio

sound(synthesizedSound,fs);

% Comprobar si el mensaje es Note Off o Note On con
elseif strcmp(msg.Type, 'NoteOff') || (strcmp(msg.Type,
'NoteOn') && msg.Velocity == 0)

    isNoteOn = false;

    note = msg.Note;

    fprintf('Note Off: %d\n', note);

    % Detener la reproducción si es necesario

    audioOut(zeros(fs * duration, 1));

end end

end

% Pausar un momento para evitar sobrecargar la CPU

pause(0.01);

end

```

Nota: Código de elaboración Propia.

Código fuente 2:

Script extractAndSaveAudioData para la extracción de la información de muestras de audio.

```

function extractAndSaveAudioData(audioFolder, outputFile)

% Obtener la lista de archivos en la carpeta especificada

files = dir(fullfile(audioFolder, '*.wav')); % Asumiendo archivos .wav

% Inicializar las variables para almacenar los datos

noteNumbers = [];

maxNoteNumber = 140; % Nota más alta en el rango dado

audioData = cell(1, maxNoteNumber); % Crear una celda para cada nota

sampleRates = NaN(1, maxNoteNumber); % Frecuencias de muestreo para

```

```

cada nota

% Procesar cada archivo

for i = 1:length(files)

    % Obtener el nombre del archivo

    fileName = files(i).name;

    % Extraer el número de nota MIDI del nombre del archivo

    [~, name, ~] = fileparts(fileName); % Obtener el nombre sin la
extensión

    noteNumber = str2double(name(1:end-1)); % Extraer el número de
nota, asumiendo que el número está al principio y es seguido por una letra
(por ejemplo, '64M')

    % Leer el archivo de audio

    [audio, fs] = audioread(fullfile(audioFolder, fileName));

    % Almacenar los datos en la celda correspondiente

    if noteNumber >= 1 && noteNumber <= maxNoteNumber

        audioData{noteNumber} = audio; % Almacenar el audio en la celda
correspondiente a la nota

        sampleRates(noteNumber) = fs; % Almacenar la frecuencia de
muestreo correspondiente a la nota

    else

        noteNumber);

        warning('Número de nota %d fuera del rango esperado.',
end

end

% Guardar los datos en un archivo .mat

save(outputFile, 'noteNumbers', 'audioData', 'sampleRates');

fprintf('Datos guardados en %s\n', outputFile);

end

```

Nota: Código de elaboración propia.

Código fuente 3.

Script processAndStoreFormants para el almacenamiento de la información relevante de los formantes de la señal de audio analizada.

```
function processAndStoreFormants(audioFolder, freqFile, ampFile, phaseFile)

    % Obtener la lista de archivos en la carpeta especificada

    files = dir(fullfile(audioFolder, '*.wav')); % Asumiendo archivos .wav

    % Inicializar variables para almacenar datos

    allFrequencies = [];

    allAmplitudes = [];

    allPhases = [];

    noteNumbers = [];

    % Procesar cada archivo

    for i = 1:length(files)

        % Obtener el nombre del archivo

        fileName = files(i).name;

        % Extraer el número de nota MIDI del nombre del archivo

        [~, name, ~] = fileparts(fileName); % Obtener el nombre sin la
extensión

        noteNumber = str2double(name(1:end-1)); % Extraer el número de
nota, asumiendo que el número está al principio y es seguido por una letra
(por ejemplo, '64M')

        % Verificar si el número de nota es válido

        if isnan(noteNumber)

            warning('Nombre de archivo no válido: %s', fileName);

            continue;

        end

        %% Extraer parciales del archivo de audio

        [frequencies, amplitudes, phases] =
extractPartials(fullfile(audioFolder, fileName), 30, 1000);

        %% Almacenar los datos

        allFrequencies{noteNumber} = frequencies;

        allAmplitudes{noteNumber} = amplitudes;
```

```

    allPhases{noteNumber} = phases;

    noteNumbers = [noteNumbers; noteNumber];

end

%% Guardar los datos en archivos .mat

save(freqFile, 'allFrecuencias', 'noteNumbers');

save(ampFile, 'allAmplitudes', 'noteNumbers');

save(phaseFile, 'allPhases', 'noteNumbers');

%comunicar resultados

fprintf('Datos guardados:\n');

fprintf('Frecuencias en %s\n', freqFile);

fprintf('Amplitudes en %s\n', ampFile);

fprintf('Fases en %s\n', phaseFile);

end

```

Nota: Código de elaboración propia.

Código fuente 4.

Función `extractPartials` para la extracción de la información de amplitud, frecuencia y fase de los parciales de una señal de audio.

```

function [frequencies, amplitudes, phases] = extractPartials(filename, threshold,
minDistance)

    % Leer el archivo de audio

    [y, fs] = audioread(filename);

    % Aplicar la transformada de Fourier

    N = length(y);

    Y = fft(y);

    f = (0:N-1) * (fs / N);

```

```

% Obtener el espectro de magnitudes y fases

magnitude = abs(Y);

phase = angle(Y);


% Solo considerar la mitad positiva del espectro

magnitude = magnitude(1:floor(N/2));

f = f(1:floor(N/2));

phase = phase(1:floor(N/2));


% Identificar los parciales (picos en el espectro)

[pks, locs] = findpeaks(magnitude, 'MinPeakHeight', threshold, 'MinPeakDistance',
minDistance);


% Extraer frecuencias, amplitudes y fases correspondientes

frequencies = f(locs);

amplitudes = pks;

amplitudes=amplitudes/max(amplitudes);

phases = phase(locs);


end

```

Nota: Código de elaboración propia.

Código fuente 5:

Función `extractAndSaveEnvelopes` para la extracción de la información de amplitud, frecuencia y fase de los parciales de una señal de audio.

```

function extractAndSaveEnvelopes(audioFolder, outputFile)

    % Obtener la lista de archivos en la carpeta especificada

    files = dir(fullfile(audioFolder, '*.wav')); % Asumiendo archivos .wav

    % Inicializar variables para almacenar las envolventes

    maxNoteNumber = 92; % Nota más alta en el rango dado

    envelopes = cell(1, maxNoteNumber); % Crear una celda para cada nota

    sampleRates = NaN(1, maxNoteNumber); % Frecuencias de muestreo para cada nota

    % Procesar cada archivo

    for i = 1:length(files)

        % Obtener el nombre del archivo

        fileName = files(i).name;

        % Extraer el número de nota MIDI del nombre del archivo

        [~, name, ~] = fileparts(fileName); % Obtener el nombre sin la extensión

        noteNumber = str2double(name(1:end-1)); % Extraer el número de nota, asumiendo que el número está al
        principio y es seguido por una letra (por ejemplo, '64M')

        % Leer el archivo de audio

        [audio, fs] = audioread(fullfile(audioFolder, fileName));

        % Convertir a mono si es necesario

        if size(audio, 2) > 1

            audio = mean(audio, 2); % Promedio de los canales para convertir a mono

        end

        % Extraer la envolvente usando la transformada de Hilbert

        analyticSignal = hilbert(audio);

        envelope = abs(analyticSignal);

        % Almacenar la envolvente en la celda correspondiente
    end
end

```



```

if noteNumber >= 1 && noteNumber <= maxNoteNumber

    envelopes{noteNumber} = envelope; % Almacenar la envolvente en la celda correspondiente a la nota

    sampleRates(noteNumber) = fs; % Almacenar la frecuencia de muestreo correspondiente a la nota

else

    warning('Número de nota %d fuera del rango esperado.', noteNumber);

end

end

% Guardar las envolventes en un archivo .mat

save(outputFile, 'envelopes', 'sampleRates');

fprintf('Envolventes guardadas en %s\n', outputFile);

end

```

Nota: Código de elaboración propia.

Código fuente 6

Función `synthesizeSoundFromData` para la síntesis de sonido a partir de los datos necesarios extraídos de las muestras.

```

function [y,frequencies,phases,amplitudes] = synthesizeSoundFromData(note, amplitude, duration, fs, matFile,
amplitudes,phases,frequencies,envelopes)

% Cargar datos del archivo .mat

data = load(matFile);

% Verificar si el número de nota está dentro del rango de datos

if note >= 1 && note <= length(data.audioData)

    audio = data.audioData{note}; % Obtener el audio correspondiente

    origFs = data.sampleRates(note); % Obtener la frecuencia de muestreo

else

    error('No se encontraron datos para la nota MIDI %d.', note);

end

```

```

% Convertir a mono si es necesario

if size(audio, 2) > 1

    audio = mean(audio, 2); % Promedio de los canales para convertir a mono

end

t = (0:length(audio)-1) / origFs;

% Re-muestrea el audio si es necesario

if origFs ~= fs

    audio = resample(audio, fs, origFs);

    fprintf('yata.\n')

end

% Extraer envolvente usando la transformada de Hilbert

envelope = envelopes(note);

% Cargar los datos desde archivos .mat

F = frequencies(note);

A = amplitudes(note);

P = phases(note);

% Ajustar la duración del sonido

numSamples = round(duration * fs);

t = (0:numSamples-1)' / fs;

% Inicializar la señal sintetizada

y = zeros(numSamples, 1);

% Generar la señal sintetizada usando un rango de frecuencias

for i=1:length(F)

    y = y + A(i)*sin(2*pi*F(i)*t + P(i)); end

% Aplicar la envolvente

% Asegurarse de que la envolvente tiene el tamaño correcto

```

```

if length(envelope) > numSamples

    envelope = envelope(1:numSamples); % Ajustar el tamaño de la envolvente

end

y = y.*envelope;

%Asegurarse de que la señal sea de una sola dimensión (mono)

if size(y, 2) > 1

    y = y(:, 1); % Convertir a mono si es necesario

end

end

```

Nota: Código de elaboración propia.

Código fuente 7.

Código de sintetizado de nota de timple mediante síntesis aditiva a partir de datos extraídos de muestras por las funciones anteriores.

```

%%LIMPIEZA DE VARIABLES

clear

clc

%%VARIABLES

envelopesdata=load('EnvelopesData.mat');

envelopes = envelopesdata.envelopes;

loadedPhaseData = load('phaseData.mat');

phases = loadedPhaseData.allPhases;

loadedAmpData = load('ampData.mat');

amplitudes = loadedAmpData.allAmplitudes;

loadedFreqData = load('freqData.mat');

```

```

frequencies = loadedFreqData.allFrequencies;

matFile="AudioData.mat";

%% Listar dispositivos MIDI disponibles

midiDevices = mididevinfo;

noteNumbers = loadedFreqData.noteNumbers;

if isempty(midiDevices.input)

    error("No se encontraron dispositivos MIDI de entrada.");

else

    % Mostrar lista de dispositivos MIDI de entrada disponibles

    fprintf('Dispositivos MIDI de entrada disponibles:\n');

    for i = 1:length(midiDevices.input)

        fprintf('%d: %s\n', i, midiDevices.input(i).Name);

    end

    % Solicitar al usuario que seleccione un dispositivo MIDI

    selectedDeviceIndex = input("Seleccione un dispositivo MIDI (por número): ");

    if selectedDeviceIndex < 1 || selectedDeviceIndex > length(midiDevices.input)

        error('Selección inválida. Por favor, seleccione un número válido de la lista.');

    end

    % Seleccionar el dispositivo MIDI elegido por el usuario

    selectedDeviceName = midiDevices.input(selectedDeviceIndex).Name;

    fprintf('Utilizando el dispositivo MIDI: %s\n', selectedDeviceName);

    % Crear objeto para recibir mensajes MIDI

    midiIn = mididevice(selectedDeviceName);

end

```

```

%% Crear objeto para reproducir sonido

fs = 44100; % Frecuencia de muestreo

audioOut = audioDeviceWriter('SampleRate', fs);

%% Inicializar variables

isNoteOn = false;

note = 0;

frequency = 0;

amplitude = 1;

duration = 1; % Duración de la nota en segundos

fprintf('Presiona una tecla en tu controlador MIDI.\n');

%% Función para convertir número de nota MIDI a frecuencia

midiToFreq = @(note) 440 * 2.^((note - 69) / 12);

while true

    % Leer mensajes MIDI

    midiMsg = midireceive(midiIn);

    % Procesar mensajes MIDI si hay alguno

    if ~isempty(midiMsg)

        for i = 1:length(midiMsg)

            msg = midiMsg(i);

            % Comprobar si el mensaje es Note On

            if strcmp(msg.Type, 'NoteOn') && msg.Velocity > 0

                isNoteOn = true;

                note = msg.Note;

                frequency = midiToFreq(note);

                amplitude = msg.Velocity / 127; % Normalizar la velocidad a [0, 1]

```

```

fprintf('Note On: %d, Frequency: %.2f Hz, Velocity: %d\n', note, frequency, msg.Velocity);

% Generar y reproducir sonido sintetizado

[synthesizedSound] = synthesizeSoundFromData(note, amplitude, duration, fs, matFile,
amplitudes,phases,frequencies,envelopes);

% Enviar el sonido al dispositivo de audio

sound(synthesizedSound,fs);

audiowrite('audio5.wav',synthesizedSound, fs);

% Comprobar si el mensaje es Note Off o Note On con Velocity 0

elseif strcmp(msg.Type, 'NoteOff') || (strcmp(msg.Type, 'NoteOn') && msg.Velocity == 0)

    isNoteOn = false;

    note = msg.Note;

    fprintf('Note Off: %d\n', note);

    % Detener la reproducción si es necesario

    audioOut(zeros(fs * duration, 1));

end

end

end

% Pausar un momento para evitar sobrecargar la CPU

pause(0.01);

end

```

Nota: Código de elaboración propia.

Código fuente 8.

Código de sintetizado de fragmento de notas de timple extraído de archivo .mid mediante síntesis aditiva a partir de datos extraídos de muestras por las funciones anteriores.

```

clear

clc

% Leer archivo MIDI

midiFile = '/Users/nestorluismesa/Documents/prueba.mid'; % Cambia esto por la ruta a tu archivo MIDI

midiData = readmidi(midiFile);

midiDevices = mididevinfo;

envelopesdata=load('EnvelopesData.mat');

envelopes = envelopesdata.envelopes;

loadedPhaseData = load('phaseData.mat');

phases = loadedPhaseData.allPhases;

loadedAmpData = load('ampData.mat');

amplitudes = loadedAmpData.allAmplitudes;

loadedFreqData = load('freqData.mat');

frequencies = loadedFreqData.allFrequencies;

noteNumbers = loadedFreqData.noteNumbers;

matFile="AudioData.mat";

startTime = 0;

fullAudio = [];

% Extraer notas

notes = midiInfo(midiData, 0);

% Mostrar información de las notas

fprintf('Notas extraídas del archivo MIDI:\n');

fprintf('Canal\tNota\tVelocidad\tInicio\tDuración\n');

for i = 1:size(notes, 1)

    fprintf('%d\t%d\t%d\t%.2f\t%.2f\n', notes(i, 1), notes(i, 3), notes(i, 4), notes(i, 5), notes(i, 6));

```

```
end
```

```
% Convertir notas MIDI a frecuencias
```

```
midiToFreq = @(note) 440 * 2.^((note - 69) / 12);
```

```
% Crear objeto para reproducir sonido
```

```
fs = 44100; % Frecuencia de muestreo
```

```
audioOut = audioDeviceWriter('SampleRate', fs);
```

```
% Generar y concatenar sonidos basados en las notas del archivo MIDI
```

```
for i = 1:10
```

```
    note = notes(i, 3);
```

```
    frequency = midiToFreq(note);
```

```
    amplitude = notes(i, 4) / 127; % Normalizar la velocidad a [0, 1]
```

```
    duration = notes(i, 6); % Duración de la nota en segundos
```

```
    fprintf('Note On: %d, Frequency: %.2f Hz, Velocity: %d\n', note, frequency, notes(i, 4));
```

```
% Generar sonido sintetizado
```

```
    [synthesizedSound] = synthesizeSoundFromData(note, amplitude, duration, fs, matFile,  
    amplitudes, phases, frequencies, envelopes);
```

```
% Crear un vector de ceros para el tiempo de silencio entre notas (si es necesario)
```

```
    silenceDuration = notes(i, 5) - startTime; % Duración del silencio antes de la nota actual
```

```
    if silenceDuration > 0
```

```
        silence = zeros(round(silenceDuration * fs), 1);
```

```
        fullAudio = [fullAudio; silence]; % Añadir silencio al audio concatenado
```

```
    end
```

```
% Añadir el sonido sintetizado al audio concatenado
```

```
    fullAudio = [fullAudio; synthesizedSound];
```



```
% Actualizar el tiempo de inicio para la próxima nota

startTime = notes(i, 5) + duration;

end

outputFileName = 'NotasConcatenadas.wav';

audiowrite(outputFileName, fullAudio, fs);
```

Nota: Código de elaboración propia.