LAB2PART1

% Get the Info

[sig, Fs] = audioread('');

fprintf('The frequency Sample is: %d \n', Fs);

% FFT

fft\_sig = fft(sig);

% Calculate Info

n = length(sig);

f = (0:n-1)\*(Fs/n);

magnitude = abs(fft\_sig);

% One-sided spectrum

half\_n = ceil(n/2);

f = f(1:half\_n);

magnitude = magnitude(1:half\_n);

% Draw the Spectrum - Original

figure(1);

plot(f, magnitude);

title('Spectrum of Don-Giovanni-1.wav');

xlabel('Frequency(Hz)');

ylabel('Amplitude');

% Using findpeaks() to find the peak points

[peaks, locs] = findpeaks(magnitude, f, 'SortStr', 'descend', 'NPeaks', 2);

hold on;

plot(locs, peaks, 'r\*', 'MarkerSize', 10);

hold off;

fprintf('The first noise frequency is: %.2f Hz\n', locs(1));

fprintf('The second noise frequency is: %.2f Hz\n', locs(2));

% Draw the semilogx - Original

figure(2);

semilogx(f, 20\*log10(magnitude));

title('Single-Sided Power Spectrum of Signal');

xlabel('Frequency (Hz)');

ylabel('Power/Frequency (dB/Hz)');

% Filter parameters

Q = 35;

% Design the notch filters using fdesign.notch and design functions

d1 = fdesign.notch('N,F0,Q', 2, locs(1), Q, Fs);

Hd1 = design(d1, 'butter');

[bn1, an1] = tf(Hd1);

d2 = fdesign.notch('N,F0,Q', 2, locs(2), Q, Fs);

Hd2 = design(d2, 'butter');

[bn2, an2] = tf(Hd2);

% Apply the notch filters

filtered\_signal1 = filter(bn1, an1, sig);

filtered\_signal = filter(bn2, an2, filtered\_signal1);

% FFT of the filtered signal

fft\_filtered\_sig = fft(filtered\_signal);

magnitude\_filtered = abs(fft\_filtered\_sig(1:half\_n));

% Draw the Spectrum - After

figure(3);

plot(f, magnitude\_filtered);

title('Spectrum of Don-Giovanni-1.wav');

xlabel('Frequency(Hz)');

ylabel('Amplitude');

% Draw the semilogx - After

figure(4);

semilogx(f, 20\*log10(magnitude\_filtered));

title('Single-Sided Power Spectrum of Filtered Signal');

xlabel('Frequency (Hz)');

ylabel('Power/Frequency (dB/Hz)');

% Save the filtered signal

%audiowrite('Filtered\_Don\_Giovanni\_1.wav', filtered\_signal, Fs);

% Optional: Listen to the original and filtered signal

% sound(filtered\_signal, Fs);

**LAB2PART2**

% Get the Info

[sig, Fs] = audioread('

/ISEP-Documents/2402-2406/2-SIGNAL/LAB/Don\_Giovanni\_2.wav');

fprintf('The frequency Sample is: %d \n', Fs);

% Filter order N

N\_values = [3,11,31];

for N = N\_values

% Define FIR filter coefficient

b = ones(1, N) / N;

% For FIR filters, the denominator coefficient is 1

an = 1;

% Calculate and draw impulse response

figure;

impz(b, an);

title(['Impulse Response - Filter Order N = ', num2str(N)]);

% Calculate frequency response

[H, w] = freqz(b, an, 'half', 1024);

% fprintf('The frequency response is: %.2f \n', w);

% Plot magnitude response

figure;

plot(w, 20 \* log10(abs(H)));

title(['Magnitude Response - Filter Order N =', num2str(N)]);

xlabel('Frequency (Hz)');

ylabel('Magnitude(dB)');

% Plot phase response

figure;

plot(w, unwrap(angle(H)));

title(['Phase Response - Filter Order N=', num2str(N)]);

xlabel('Frequency (Hz)');

ylabel('Phase (radians)');

end

**LAB3PART1**

% Get the Info

[sig, Fs] = audioread('

/ISEP-Documents/2402-2406/2-SIGNAL/LAB/LAB3/Pa11.wav');

%sound(sig, Fs);

fprintf('The frequency of Pa11.wav is: %d Hz\n', Fs);

% Calculate power spectral density

Y = fft(sig);

PSD = abs(Y).^2;

% Inverse Fourier Transform to obtain the autocorrelation function

R\_fft = ifft(PSD);

% Use the Matlab function xcorr

[R\_xcorr, lags] = xcorr(sig, 'biased');

% Plot two autocorrelation functions

figure;

subplot(2,1,1);

plot(real(R\_fft));

title('Autocorrelation Functions - Inverse FFT');

subplot(2,1,2);

plot(lags, R\_xcorr);

title('Autocorrelation Functions - xcorr');

% Separating positive delay values from autocorrelation functions

% Identify echo delay using only the positive lags

positiveLags = lags(lags >= 0);

positiveR\_xcorr = R\_xcorr(lags >= 0);

% Assume minLag as some small fraction of signal length to avoid direct signal peak

minLag = round(0.001 \* length(sig));

% Using findpeaks() to find the peak points

% Find peaks with minimum peak height to avoid detecting noise as echo

[pks, locs] = findpeaks(positiveR\_xcorr, 'MinPeakHeight', max(positiveR\_xcorr)/4, 'MinPeakDistance', minLag);

fprintf('Number of peaks found: %d\n', length(pks));

% From the result can see that there are 2 points

% Plot the peaks on the autocorrelation function

figure;

subplot(2,1,1);

plot(positiveLags, positiveR\_xcorr);

title('Autocorrelation Function - xcorr (Positive Lags)');

hold on;

plot(positiveLags(locs), pks, 'r\*', 'MarkerSize', 10);

hold off;

% ===Design filters===

% For two echo points, the original audio can be expressed as:

% x(n)=s(n)+α⋅s(n−D1)+β⋅s(n−D2)

% Filters : h(n)=δ(n)−α⋅δ(n−D1)−β⋅δ(n−D2)

% y(n)=x(n)∗h(n)

% Assume echo\_delay as the location of the first peak detected

% The first peak after zero lag

echo\_delay = positiveLags(locs(1));

second\_echo\_delay = positiveLags(locs(2));

fprintf('Estimated echo delay is: %d samples\n', echo\_delay);

% Assume echo attenuation

% Since there are two peak points, two parameters are designed here

alpha = 0.6;

beta = 0.3

% Design a filter to remove a single echo: h(n)

filter\_length = max(echo\_delay, second\_echo\_delay) + 1;

filter = zeros(filter\_length, 1);

% Direct signal component

filter(1) = 1;

% Echo signal component

filter(echo\_delay + 1) = -alpha;

filter(second\_echo\_delay + 1) = -beta;

% Apply the filter to remove the echo

% Use 'full' for convolution

y\_filtered = conv(sig, filter, 'full');

% Trim the filtered signal to the original length

y\_filtered = y\_filtered(1:length(sig));

% Play the signal after echo removal

sound(y\_filtered, Fs);

% Plot the original and filtered signals for comparison

figure;

subplot(2, 1, 1);

plot(sig);

title('Original Audio Signal');

xlabel('Sample Number');

ylabel('Amplitude');

subplot(2, 1, 2);

plot(y\_filtered);

title('Audio Signal After Echo Removal');

xlabel('Sample Number');

ylabel('Amplitude');

**LAB3PART2**

% Get the Info

[sig, Fs] = audioread('

/ISEP-Documents/2402-2406/2-SIGNAL/LAB/LAB3/Pa11.wav');

% Perform an FFT on the signal and generate a frequency vector

N = length(sig);

Y = fft(sig);

F = linspace(0, Fs, N);

% Spectrogram of the original signal

figure;

plot(F, abs(Y));

title('Spectrum of Original Audio Signal');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

grid on;

hold on;

for k = 1:7

xline(k \* Fs / 8, 'r--');

end

hold off;

% Apply mapping rules for exchange

% Generate index mapping array swapIndices for swapping frequency components

swapIndices = 1:N;

for k = 2:(N/2)

% Exclude the Nyquist frequency point, which is at position N/2+1

if k ~= N/4+1

swapIndices(k) = N-k+2;

swapIndices(N-k+2) = k;

end

end

Y\_swapped = Y(swapIndices);

% Apply IFFT to obtain the corrected signal

correctedSig = real(ifft(Y\_swapped));

% Calculate the FFT of the corrected signal

Y\_corrected = fft(correctedSig);

% Spectrogram of the corrected signal

figure;

plot(F, abs(Y\_corrected));

title('Spectrum of Corrected Audio Signal');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

grid on;

hold on;

for k = 1:7

xline(k \* Fs / 8, 'r--');

end

hold off;

% Since the difference cannot be directly seen by observing the two images

% the following operations are performed:

% Calculate the difference between the spectra of two signals

magnitude\_diff = abs(Y) - abs(Y\_corrected);

% Spectral difference plot

figure;

plot(F, magnitude\_diff);

title('Magnitude Difference Between Original and Corrected Spectrum');

xlabel('Frequency (Hz)');

ylabel('Magnitude Difference');

grid on;

hold on;

for k = 1:7

xline(k \* Fs / 8, 'r--');

end

hold off;

**LAB4**

%% Value Settings

% Set the amplitude value 1 for the first smoothing process (remove background noise)

threshold1 = 0.4;

% Set the amplitude value to 1 for smoothing again (remove dial tone)

threshold2 = 1.0;

%% Read audio files

%% No.0

% filename = '

/ISEP-Documents/2402-2406/2-SIGNAL/LAB/LAB4/0123456789.wav';

% The audio '0123456789.wav' dial is: 0123456789

% Set threshold1 = 0.2 & threshold2 = 0.6;

[y, Fs] = audioread(filename);

% fprintf('The sampling frequency is: %d Hz\n', Fs);

%% Plot original signal

t = (0:length(y)-1)/Fs;

figure;

plot(t, y);

title('Original Tone (time domain)');

xlabel('Time (seconds)');

ylabel('Amplitude');

%% Design a high-pass filter with a cutoff frequency of 650 Hz

% Because the lowest frequency of key tone is 697 (1.5% error tolerance rate)

cutoff\_freq = 650;

[b, a] = butter(10, cutoff\_freq/(Fs/2), 'high');

% Apply a high-pass filter to the audio signal

y\_filtered = filter(b, a, y);

y\_filtered\_smooth1 = y\_filtered;

y\_filtered\_smooth1(abs(y\_filtered) < threshold1) = 0;

% Plot the filtered and smoothed signal

figure;

plot(t, y\_filtered\_smooth1, 'Color', [1, 0.5, 0]); % set to orange

title('Filtered & Smoothed Tone - Background Noise Removal (time domain)');

xlabel('Time (seconds)');

ylabel('Amplitude');

%% Smooth the filtered signal to remove background noise

y\_filtered\_smooth = y\_filtered\_smooth1;

y\_filtered\_smooth(abs(y\_filtered\_smooth1) < threshold2) = 0;

% Plot the filtered and smoothed signal

figure;

plot(t, y\_filtered\_smooth, 'Color', [1, 0, 0]);

title('Filtered & Smoothed Tone - Dial Tone Removal (time domain)');

xlabel('Time (seconds)');

ylabel('Amplitude');

%% DTMF frequency and key mapping

dtmf\_freqs = [697, 770, 852, 941, 1209, 1336, 1477, 1633];

dtmf\_keys = [

'1', '2', '3', 'A';

'4', '5', '6', 'B';

'7', '8', '9', 'C';

'\*', '0', '#', 'D'

];

low\_freqs = [697, 770, 852, 941];

high\_freqs = [1209, 1336, 1477, 1633];

freq\_tolerance = 0.015; % fault tolerance

%% Audio Segmentation

% Defines the threshold for silent segments (amplitude is 0 for 80ms continuously)

silent\_duration\_threshold = 0.08;

silent\_sample\_threshold = round(silent\_duration\_threshold \* Fs);

% Traverse the signal to detect valid sound segments

effective\_tones = [];

start\_idx = 1;

in\_silent = false;

silent\_start\_idx = -1;

for i = 1:length(y\_filtered\_smooth)

if abs(y\_filtered\_smooth(i)) == 0

if ~in\_silent

silent\_start\_idx = i;

end

in\_silent = true;

else

if in\_silent && (i - silent\_start\_idx >= silent\_sample\_threshold)

if start\_idx < silent\_start\_idx

effective\_tones = [effective\_tones; start\_idx, silent\_start\_idx - 1];

end

start\_idx = i;

end

in\_silent = false;

end

end

% Check: if the last segment is a valid sound segment, add it to the list

if ~in\_silent && start\_idx < length(y\_filtered\_smooth)

effective\_tones = [effective\_tones; start\_idx, length(y\_filtered\_smooth)];

elseif in\_silent && (length(y\_filtered\_smooth) - silent\_start\_idx >= silent\_sample\_threshold)

effective\_tones = [effective\_tones; start\_idx, silent\_start\_idx - 1];

end

% Print the dialed number (number digits)

% fprintf('Total number of dialed digits: %d\n', size(effective\_tones, 1));

fprintf('The number dialed is: ');

%% Obtain the main frequency information of each valid sound segment and compare it

for i = 1:size(effective\_tones, 1)

segment = y\_filtered\_smooth(effective\_tones(i, 1):effective\_tones(i, 2));

if isempty(segment)

continue;

end

Y\_segment = fft(segment);

L\_segment = length(segment);

P2\_segment = abs(Y\_segment / L\_segment);

P1\_segment = P2\_segment(1:floor(L\_segment/2)+1); % Make sure to use integer indexing

P1\_segment(2:end-1) = 2 \* P1\_segment(2:end-1);

f\_segment = Fs \* (0:(floor(L\_segment/2))) / L\_segment;

% Find the two main frequency components

[sorted\_amplitudes, indices] = sort(P1\_segment, 'descend');

main\_freqs = sort(f\_segment(indices(1:2))); % Sort by frequency

% Find the closest DTMF frequency

[~, low\_idx] = min(abs(low\_freqs - main\_freqs(1)));

[~, high\_idx] = min(abs(high\_freqs - main\_freqs(2)));

% Make sure the frequency is within the allowable error range

if abs(low\_freqs(low\_idx) - main\_freqs(1)) / low\_freqs(low\_idx) <= freq\_tolerance && ...

abs(high\_freqs(high\_idx) - main\_freqs(2)) / high\_freqs(high\_idx) <= freq\_tolerance

% Output the corresponding button

detected\_key = dtmf\_keys(low\_idx, high\_idx);

fprintf('%s', detected\_key);

else

fprintf('The NO. %d key cannot be recognized\n', i);

end

end

fprintf('\n ');

%% Plot filtered and smoothed signals and valid sound clips

figure;

plot(t, y\_filtered\_smooth, 'Color', [1, 0, 0]);

hold on;

for i = 1:size(effective\_tones, 1)

start\_time = (effective\_tones(i, 1) - 1) / Fs;

end\_time = (effective\_tones(i, 2) - 1) / Fs;

fill([start\_time, end\_time, end\_time, start\_time], [min(y\_filtered\_smooth), min(y\_filtered\_smooth), max(y\_filtered\_smooth), max(y\_filtered\_smooth)], 'g', 'FaceAlpha', 0.3, 'EdgeColor', 'none');

end

title('Effective Sound Area (time domain)');

xlabel('Time(secoonds)');

ylabel('Amplitude');

hold off;

**LAB5**

%% Read the image

B = imread('

/ISEP-Documents/2402-2406/2-SIGNAL/LAB/LAB5/fichier2.bmp', 'bmp');

B = 255 \* B;

figure, imshow(B, []);

colormap(gray);

title('Original Image');

%% Pixels shifted calculation

% Initialize the array storing the shift

offsets = zeros(size(B, 1), 1);

% Calculate the offset of each row relative to the adjacent previous row

for i = 2:size(B, 1)

previous\_row = double(B(i-1, :));

current\_row = double(B(i, :));

% Use the xcorr function to calculate the cross-correlation between two adjacent rows

[correlation, lags] = xcorr(previous\_row, current\_row);

% Find the offset corresponding to the maximum correlation value

[~, max\_index] = max(correlation);

offsets(i) = lags(max\_index);

end

% Accumulate offsets so that all rows are aligned relative to the middle row

mid\_index = floor(size(B, 1) / 2);

cumulative\_offsets = cumsum(offsets);

shift\_to\_middle = cumulative\_offsets(mid\_index);

%% Generate correct image

% Create a new image matrix

corrected\_B = zeros(size(B), 'like', B);

% Adjust the position of each row so that it is aligned relative to the middle row

for i = 1:size(B, 1)

shift\_amount = cumulative\_offsets(i) - shift\_to\_middle;

corrected\_B(i, :) = circshift(B(i, :), [0, shift\_amount]);

end

% Convert image data to 8-bit unsigned integer type

corrected\_B = uint8(corrected\_B);

% Display the recovered image

figure, imshow(corrected\_B, []);

colormap(gray);

title('Correct Image');